DATA FARMING

for simulation analysis

Susan M. Sanchez and Paul J. Sanchez

DRAFT VERSION 0.99.9: July 2025

Copyright © 2025 by Susan M. Sanchez and Paul J. Sanchez
All rights reserved
License info coming soon....
ISBN 978-0-9967784-4-2

Table of Contents (click to navigate)

Chapter 1

Introduction

Key Concepts

- Data farming is a process for gaining understanding and insight. Designed experimentation provides a roadmap for systematically exploring broad system behaviors.
- Data farming is iterative. Embedding it throughout the modeling and analysis processes promotes both model improvements and analyst insights.
- Data farming facilitates discussion between different stakeholders by identifying and quantifying tradeoffs between competing viewpoints.

We live in a world inundated with data. The term 'data mining' is ubiquitous in the literature, while 'data analytics' and 'data science' have skyrocketed in popularity in recent years. Much of these large volumes of data is collected automatically—by our communication technology, sensors in the environment, cookies placed on websites, wireless devices comprising the internet of things, and more.

Many simulation models can benefit from data mining. Some of these observational data sources can be used to characterize input distributions for stochastic simulation models, by fitting distributions from which pseudo-random numbers are generated, by bootstrapping samples from empirical distributions, or by using it in data-driven simulation models or digital twins to affect real-time system intervention and control. This can be helpful for understanding complex systems that already exist. Yet this type of data is observational by nature, and so has limitations. It is of limited use to help us understand the behavior of systems that do not yet exist, and hence for which there is no body of existing data to mine. Studying prospective systems to determine their behavior and utility is an important aspect of simulation modeling. In this case, a simulation model becomes a *source* of data we can use to develop insights and improve decision making rather than a *consumer* of pre-existing data.

1.1 Data Farming: The Metaphor

Consider this description from Sanchez (2018):

> Real-world farmers cultivate the land to maximize their yield. They manipulate the environment to their advantage by using irrigation, pest control, crop rotation, fertilizer, and more. Small-scale designed experiments can help them to determine whether these treatments are effective. Similarly, data farmers manipulate simulation models to their advantage—but using large-scale designed experimentation. This allows them to learn more about the simulation model's behavior in a structured way. In this fashion, they 'grow' data from their models, but in a manner that facilitates identifying the useful information. For large-scale simulation experiments, this often results in data sets that, while big, are far smaller than what would be needed to gain insights if the results were observational (i.e., obtained using ad hoc or randomly generated combinations of factor settings). Data generated prospectively from designs is also better, in the sense that it lets us identify root cause-and-effect relationships between the simulation model input factors and the simulation output.

Simulation is not the only community to use the data farming metaphor. As a noun, a "data farm" may refer to a large bank of connected computers used to process and store data, host web services, provide access for scientific computing, and more. As a verb, data farming has been used as a metaphor for dealing with big data in non-simulation contexts: see, e.g., Kusiak (2006) for enhancing industrial data for decision-support purposes, or Mayo et al. (2016) for improving patient outcomes in healthcare settings. These data farming approaches attempt to improve the collection, storage, maintenance, and retrieval of observational data so it is faster and easier to harvest insights. While some effort has been made to address causality from observational datasets (Pearl 2009), we can distinguish the simulation data farming viewpoint as one of generating and analyzing *inferential* big data, in contrast to methods for curating and analyzing *observational* big data.

Schruben (2017) asserts that "model is a verb" for simulation professionals. Likewise, data farming is a verb from the simulation perspective used in this book.

1.1.1 Terminology

Factors are inputs (or functions of inputs) to a simulation model that are purposefully varied at different levels when growing the data from a simulation experiment.

An *experiment design* for k factors is an $n \times k$ matrix or table where each of the k columns specifies the levels or settings for a single factor and each of the n rows specifies the combination of factor settings that define a particular scenario. We refer to the rows as *design points*. They might also be called *runs* or *trials* in the general statistics literature.

Features are characteristics of the response surface—the implicit mapping between the simulation model's inputs to their corresponding outputs. A statistical or analytical model of this I/O behavior is called a *metamodel*—it is a mathematical model of the computational model. Many types of metamodels are possible, including partition trees (also known as classification and regression trees), multiple regression metamodels, logistic regression metamodels, Gaussian process metamodels, neural networks, random forests, and more.

Flexibility is an important consideration when embarking on a data farming study because the types of designs used to grow the data will affect the types of metamodels we can fit, and consequently the types of questions we can answer. In the data farming context, we are proponents of 'thinking big' in terms of the number and types of factors, the number of outputs and breadth of their response surface behaviors, and the types of analysis tools and methods that can be applied to the output data.

1.1.2 The Data Farming Process

Figure 1.1 illustrates our recommended structure for the data farming process. We have successfully used this iterative approach for over two decades as a basis for thesis work, academic research, and sponsored projects. It incorporates three interconnected "Plan–Do–Study–Act" (PDSA) cycles. The PDSA acronym was coined by W. Edwards Deming in 1986, though he had proposed this process for continuous improvement in manufacturing going back to the early 1950's (Moen, 2020). In the data farming context, there are three distinct cycles: the modeling cycle, the insight cycle, and the impact cycle. We can think of the PLAN steps as deciding what next to address via coding or computational experiments, the DO steps as "growing" data from our computational models, and the STUDY steps as

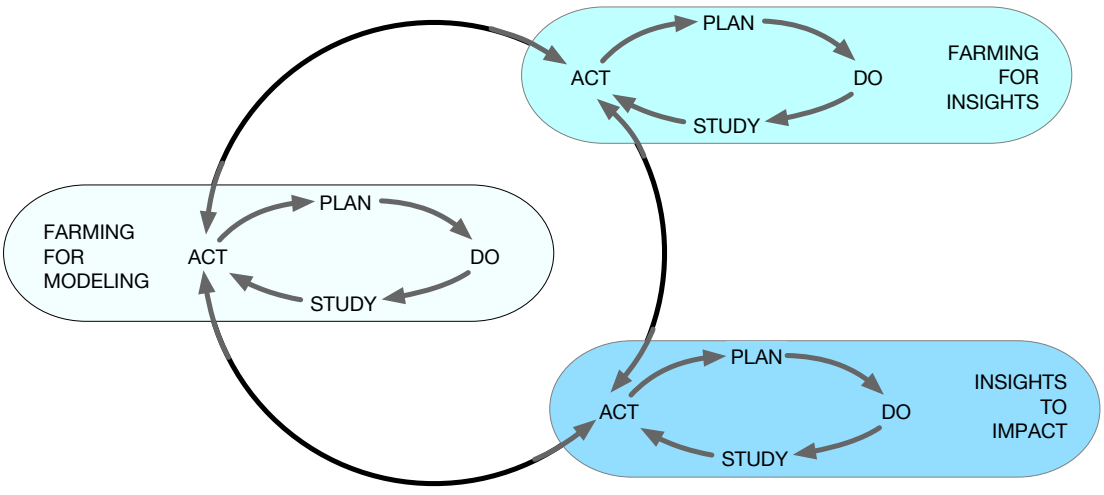


Figure 1.1: Data Farming Process

using statistical and visualization techniques to reap insights about the model's behavior. In the ACT steps, depending on what the analysis and visualization reveal, we can either (i) continue to iterate within the same cycle, or (ii) move to a different cycle and cross-communicate the actions with those team members and stakeholders responsible for that cycle's PDSA process.

Note that other formalizations can be found in the literature. One example is the "loop of loops" construct in NATO (2014), which also represents an iterative, collaborative process connecting the realms of model development, rapid scenario prototyping, design of experiments, high performance computing, and analysis & visualization. Another is the knowledge discovery in simulation data (KDSD) process described in Feldkamp et al. (2020a). See Hunker (2024) for a more detailed discussion of different formalizations.

In this book, we focus on the "farming for insights" cycle. We provide tested simulation models that the self-directed learner can use to gain hands-on experience, or the instructor can use to create assignments in addition to any other models that they wish to investigate. However, we emphasize that for those building simulation models, our experience is that incorporating data farming during model development dramatically reduces the time required to obtain a functioning model and improves outcomes. It should *not* be restricted to the end stages of a study! It is most effective when used throughout the entirety of the modeling and analysis process.

A more detailed discussion of the cycles in Figure 1.1 follows.

Start

- Begin with a question or problem of interest. This can be a broad question: a decision maker is more likely to be interested in "How can I improve the performance of my hospital?" than "What is the average waiting time for patients arriving at the emergency department?" The cycle in which to start depends, in part, on the maturity of the simulation model—but you should start with a PLAN. For example, start your PLAN in the modeling cycle if you're developing a single simulation, start your PLAN in the insights cycle if you already have a functioning model, or start your PLAN in the insights-to-impact cycle if you are getting started on a large project where you first must determine what to simulate.

Farming for modeling cycle

- PLAN: The first time through this cycle, come up with a conceptual model and determine what combination of languages or simulation packages is well-suited for implementing the model. With the premise of making the model "as simple as possible, but no simpler" (Sanchez, 2009), start small and iteratively add functionality to the model. Set up the code so it is easy to automate the data farming process. Determine the outputs of interest, as well as factors (inputs) and notional factor ranges. Choose an experiment design appropriate for verification purposes: this can be small so the runs can be made quickly.

- DO: Conduct the experiment to grow data.
- STUDY: Use analysis and visualization tools to assess whether or not the new functionality works as intended, or whether anything else breaks after adding this new functionality.
- ACT: If all appears well, confirm the code changes—this is a great place to commit revisions if you use a version control system such as git, mercurial, or SVN.
  - Continue with the modeling cycle if other features of the conceptual model await implementation.
  - Move to the insights cycle if you are ready to explore the model's behavior more broadly.
  - Move to the impact cycle if your experiment was conducted to address a decision-makers' concern about using the model insights for real-world interventions.

Large software projects can quickly become overwhelming. Using the incremental approach built into this cyclic approach helps tame the complexity, and provides tangible checkpoints where you can confirm that your target goals are being met.

Farming for insights cycle

- PLAN: Take more care in determining the outputs of interest, as well as a comprehensive set of factors you wish to vary and the factor ranges or sets of levels of interest. Choose an experiment design appropriate for broadly exploring the data and generating robust insights: this is apt to be a large-scale design.
- DO: Conduct the experiment to grow data.
- STUDY: Use analysis and visualization tools to explore how the model's key performance metrics change as functions of changes in the input factors.
- ACT: If all appears well, confirm any code changes and log any analysis findings and supporting data—this is a great place to commit revisions if you use a version control system such as git, mercurial, or SVN.
  - Move to the modeling cycle if potential bugs are uncovered, or questions arise that require additional output or enhancements to the conceptual model.
  - Stay in the insights cycle if the model appears to be working properly, but interesting questions arise that require additional experimentation, such as new factors worthy of exploration, modifications to the factor ranges, additional replication, or a different type of design.
  - Move to the impact cycle if the insights are actionable.

This cyclic process is specifically geared towards incorporating knowledge and insights as you gain them. Keep in mind what your primary goal is—whether it is to develop a basic understanding of the simulation's behavior, to find robust decisions or policies, or to compare the merits of various decisions or policies. Exploring your model, like exploring in the real world, is a dynamic process that should evolve as you learn.

Insights to impact cycle

- PLAN: Insights from this cycle form the basis upon which decision makers and other stakeholders will decide what (if any) actions to take in the real-world setting. This might involve options such as conducting a small real-world experiment to gather data for model validation purposes, or making a particular decision such as purchasing a set of resources or selecting a particular facility location or factory floor layout. Regardless, plan what data are needed to determine whether or not the interventions are successful.
- DO: Conduct the experiment or implement the interventtions.
- STUDY: Assess whether the experiment results are consistent with the data farming predictions, or whether the interventions appear beneficial.
- ACT: Document the findings so they are accessible to a broad group of stakeholders.
  - Move to the modeling cycle if results from the real-world experiment differ from that anticipated from the model-based insights. There may be some additional features of the real-world system that should be included in the conceptual model, or input data modeling assumptions might need to be changed.
  - Move to the insights cycle if the model appears to be working properly, but interesting questions arise that require additional experimentation. These might include action items such as new factors worthy of exploration, modifications to the factor ranges, additional replications, or a different type of design to enable alternative analyses. Some factors from earlier iterations may no longer be relevant once interventions are made.
  - Stay in the insights-to-impact cycle if the decision makers feel comfortable moving on to other intervention decisions without first conducting additional computational experiments, or if they are now ready to expand a pilot study to a broad-based intervention.

This cycle promotes the use of a living simulation model as part of a continuous improvement process. Stakeholders and team members iterate through model-based interventions applied to the real-world scenario, and over time they should develop more confidence in the data farming process and a better understanding of the strengths and limitations of their underlying simulation models. The process yields quicker turnaround times as familiarity and confidence grow.

1.2 Expanding the Metaphor

The data farming metaphor can be expanded beyond that of growing data, as described in this section. The topics that follow have been adapted from Sanchez (2021).

1.2.1 Cross Fertilization

In our experience, data farming is most effective when it is a collaborative effort (NATO, 2014). Stakeholders in the problem domain help keep the data farming effort well-grounded and ensure that it does, in fact, address questions and provide insights that are useful and interesting to decision makers. Problem domain experts are key players in the model validation process. Simulation modelers can contribute their expertise in a variety of ways. At early stages of a simulation study, their conceptual modeling skills may help scope the project so the simulation model is neither overly simplified nor overly complex for its intended purposes. They catch logical misconceptions that might invalidate the results or interpretation, such as: a user who does not realize that different random number seeds yield different outcomes; that modeling a queue as capacitated vs. uncapacitated will yield different results; or that replacing distributions with their means can completely invalidate a stochastic model.

One practical piece of advice is to ask all stakeholders to jot down a few key expectations, such as "What do you believe the three most important factors will be? How will they affect the response?" Done early, this may lead to discussions that help frame the conceptual model and make sure there is a common understanding of its components, especially if the stakeholders have different backgrounds and expertise. Done before running the experiment, this helps ensure that the factors, their ranges or settings, and the experimental design used will be suitable for addressing the initial questions—although it is better to think of experimentation and analysis as an iterative process rather than as a single event. Done after the data have been generated but before conducting analysis, this may help clarify whether or not the results are surprising. Ultimately, when a surprising result is found, it should either lead to a bug being fixed (model verification) or intuition being changed (model validation).

1.2.2 Sowing the Seeds

Design of experiments (DOE) can be viewed as sowing the seeds for successful data farming, and brings tremendous capabilities to simulation studies. There are several reasons for this. First, experimentation is a straightforward way of establishing cause-and-effect. By purposefully varying factors using a good design, we can observe what (if any) effects they have on the responses—at least within the context of our simulation model. Varying multiple factors simultaneously is the only way to reveal interaction effects. Varying factors at many levels in a space-filling design provides analysis flexibility, and using a good experimental design is absolutely required. What constitutes a bad design? A one-factor-at-a-time design is bad because it does not reveal any interactions. A design with high correlations among factors is bad because it means that factor effects are confounded, so there is no way to determine the individual effects of the factors on the response. A design that cannot be executed within the required time-frame is bad because it means the decision maker will not be able to draw insights from the study. A design that ignores factors simply to reduce

the number of design points is bad because it drastically limits the potential insights that can be gained—this could be disastrous if any impactful factors are omitted. A design that varies factors at only a small number of levels may be bad because it limits our ability to fit metamodels to complex response surfaces or to provide lack-of-fit indicators for these metamodels.

There are many good experimental designs, but some are more suitable for physical experiments or deterministic computer experiments than for stochastic simulation experiments. Here are a few classical designs often used in physical experimentation. They are covered more deeply in Section 3.3—not because they are well-suited for data farming studies, but because they are easy to understand and facilitate the presentation of the principles of designed experiments.

- Factorial experiments, a.k.a. gridded designs;
- Two-level fractional factorial experiments; and
- Central composite designs (non-rotatable and rotatable).

Subsection 3.3.7 presents extensions of these designs that we developed specifically for large-scale data farming experiments:

- Resolution V fractional factorials (R5FFs); and
- Rotatable and non-rotatable resolution V central composite designs (R5CCDs).

Note that these designs can handle large numbers of factors, but explore each factor at a small number of levels.

Section 3.4 fills out the list with modern space-filling designs. These are designs that we strongly recommend and use most often in our own data farming studies:

- Nearly orthogonal Latin hypercubes (NOLHs);
- Nearly orthogonal-and-balanced (NOAB) designs; and
- Frequency-based designs and screening designs (FBDs and FBSs).

As a practical tip, follow the links in Appendix A to download the software and run a data farming experiment. Use this software to easily generate any of the designs listed above.

1.2.3 Pest Control

In Section 1.2.1, we described how stakeholders' predictions of which factors will be most important can be helpful in verification and validation (V&V) efforts. A large-scale sensitivity analysis is a much broader and more rigorous way of stress-testing a simulation model.

This debugging effort also reinforces the view that model is a verb. We should not separate the process of modeling and experimentation—they enhance each other. It is better to continually experiment as you go along and build a model—catching as many bugs as you can early on—rather than waiting until the end. Experimentation can also help the modeler

avoid adding unnecessary model detail if it becomes clear that variation in certain model subcomponents is dampened by the system, so additional complexity is not warranted. For example, if varying a deterministic setup time for a station in a job shop between 15 minutes and 30 minutes does not yield a noticeable difference in overall throughput, then it would not be worthwhile to expend effort to create a stochastic setup time that varies over that same range.

At any stage, a practical way of proceeding is to begin with a baseline design point. Set the ranges for each quantitative factor a small percentage (say, 5% or 10%) above and below the baseline (if the baseline is at the lowest or highest level of interest, expand the range in only one direction). Run designed experiments regularly during the model-building process. This also means the model you're making will be data farmable, which will save you the time of having to restructure the finished model or create a data farming wrapper to facilitate experimentation. It also means you will easily identify situations where the model behaves strangely or stops working. We have often found it possible to diagnose and track down errors using this approach. For example, in one experiment we varied thirty simulation inputs that we had previously left unchanged, and found that the simulation sporadically failed to run to completion. A few splits of a partition tree isolated the problem to an interaction between two factors that could result in a buffer overflow. The bug was easy to correct once it was identified.

1.2.4 Harvesting Efficiently

Automation is a key enabler of data farming, since there are many repetitive tasks. A little work up front makes life much easier down the road.

A common misconception these days is that using the command-line/terminal is primitive, and that graphical user interfaces (GUIs) are modern and efficient. The problem with GUIs is that they require a person to be "in-the-loop". If you plan to run large-scale explorations of your model, command-line scripting is essential to automate the process. The good news is that batch processing is so repetitive that it is quite straightforward to automate, and we have written tools to handle common patterns of run-control. If you are just getting started on data farming, you may find it helpful to use some of the run control scripts in the datafarming Ruby gem described in Appendix A. These are scripts that allow you to run any simulation models that can be run from the command line (such as simulations written in python, Matlab, R, Java, C, C++, or similar languages).

When you are ready, parallel computing can easily be leveraged for purposes of data farming. Each run (a single replication of a single design point) is a self-contained simulation that can be sent off to a core, with the data consolidated once all runs are complete. Software such as HTCondor (available at `https://research.cs.wisc.edu/htcondor/`) can be used to distribute jobs to multiple cores, either on a single multicore machine or on a computing cluster. SESSL (available at `http://sessl.org`) is another software language set up to

facilitate experiments for a variety of simulation modeling platforms (Ewald and Uhrmacher, 2014; Warnke and Uhrmacher, 2018). For more about the nuts and bolts of data farming, see Sanchez and Sanchez (2017). If your models are set up to be data farmable from the start, running the data farming experiments is straightforward—and you will never want to go back to manual experimentation.

1.2.5 Maximizing Yield

By maximizing yield, we mean gaining as much knowledge and insight as we can from our simulation study to inform decision makers. This may be insight about the simulation model's behavior itself, or about a real-world situation that we are simulating. If the model's intended use is to assist decision makers on important and complex questions, then we should 'think big' in terms of the insights that might be gained. Decision makers attempting to address complex problems are not likely to be interested in answers to overly simple questions. Given the time and effort that can be spent to conceptualize and implement a simulation model, make sure that effort is put to good use. Data farming is a way to make your simulation model work for you!

Think of robustness as you plan your data farming experiments (Sanchez and Sanchez, 2020). Robustness is a structured way to guard against making unwarranted assumptions. It can also facilitate the successful transfer of model-based insights into real-world decision making. Factors in your data farming experiment can be differentiated as decision factors, noise factors, and artificial factors. Decision factors are those that can be controlled in the real-world setting for which the simulation is based. Noise factors are those that either cannot be controlled, or can be controlled only at great cost or difficulty, in the real world. Artificial factors are specific to the simulation environment, such as the warm-up period for steady-state simulations; choices of random number generators, seeds, or streams; run lengths; time intervals for discrete-time simulations; and more. Including artificial factors in a data farming experiment may yield insights about using simulation for real-time control. Including both decision and noise factors makes it more likely that recommended solutions will work well for a broad range of situations that might arise in practice, even if these are not optimal solutions for any particular setting. A robustness perspective can also be used to ascertain whether certain model assumptions, such as input distribution shapes, lead to substantively different recommendations. The current combination of computational power and modeling platforms and paradigms helps simulation modelers to reduce so-called 'Type III errors' of solving the wrong problem (Mitroff and Featheringham, 1974). Seeking robust solutions aids this process.

1.2.6 Reaping the Benefits

Once we have generated an inferential big data set from our data farming experiment, what do we do with it? We have found that just as having big data from the internet meant that companies found new and exciting things to do with it, having big data from simulation experiments offers the opportunity for new and interesting ways of looking at the results (Elmegreen et al., 2014). These include a wide variety of metamodeling and visualization techniques.

Theoretically, every one of the inputs should affect at least one response in some way. If not, there is something wrong with either the conceptual model (e.g., we have added unnecessary detail or left out connections) or its implementation (the code contains bugs). However, even if all factors have some 'true' effect, that does not mean they are all equally important. Data farming can help us identify the factors or interactions that are key drivers of performance over the region of factor exploration. Consequently, when constructing metamodels we may end up excluding factors or terms that are statistically significant—either because they are dwarfed by other factors or terms that have much stronger effects, or because their effects, while statistically significant, are not of practical interest given the region of exploration for this particular experiment.

Some features are best revealed by graph-analytic techniques: see, e.g., Feldkamp et al. (2020b), Matković et al. (2018), or Sanchez (2020) for examples drawn from simulation experiments. Past data farming studies have helped save lives, time, money, and the environment; improve algorithms; and facilitate thoughtful discussions around modeling human behaviors and interactions.

1.2.7 Sharing the Harvest

Our metaphor involves farming, not gardening. In the real world, both might be used to grow vegetables (or herbs, or flowers)—but a garden is a small plot intended for private use, while a farm is a larger enterprise that grows crops for others. This sense of distributing results to a large community of stakeholders, rather than simply generating the insights for ourselves, is important. The sense of scale also matters. We have over a trillion times the computing power at our fingertips than was used to first put a man on the moon (Lucas et al., 2015). How are we leveraging this power? Are our methods of building and analyzing simulation models keeping pace?

1.3 Why is it useful?

Our observation is that, throughout much of its history, simulation has been used descriptively. Models have been focused on describing a current or anticipated system's behavior,

but principally at one or a small handful of scenarios. When scenarios were explored, they often stimulated heated discussions but didn't necessarily give clear answers when many factors were changed simultaneously. Data farming represents a significant change of perspective. It directly addresses the question of how the system changes as a function of its inputs, and opens up the world of prescriptive rather than descriptive analytics.

With data farming, your simulation model works for you. You can implement these methods with minimal additional effort compared to a generic simulation study, but the gains can be tremendous. Data farming speeds up the process of implementing a functioning model. Model developers can also use data farming to provide guidance about the marginal value of adding model complexity. Simulation analysts can get bigger payoffs from the time and effort spent building, verifying, and validating the simulation model—not just answering a specific question or two, but potentially generating insights about worthwhile opportunities or avoidable risks that might not have been apparent prior to the data farming study. Simulation researchers can improve algorithm performance by studying the impact of algorithms' hyperparameter values, or generate output datasets in a structured way to assess how those algorithms work for different types of models. Data scientists, similarly, can study the impact of their algorithms' hyperparameters on performance, where the datasets used to drive these conclusions arise from structured experiments and so offer the possibility of learning across multiple studies. Finally, policy makers may find it particularly useful to have a virtual venue for a model-based exploration. This can change the communication dynamics among multiple stakeholders, who often have competing goals, by helping identify whether they disagree on the underlying data and assumptions or their beliefs of how those assumptions will affect responses of interest. Ideally, this will shift anecdotal arguments to more thoughtful discussions of the broad impacts of various policies or interventions, and improve the chances of arriving at consensus or reasonable compromise decisions.

1.4 Summary

The chapters that follow provide a *practical* approach to data farming that leverages several recent theoretical advances in the design of experiments field, as well as modern data analysis and visualization techniques. Going forward, there are many opportunities for advancing the theory, the practice, and the applications of simulation. This work can be worthwhile, rewarding, fascinating, and fun! We hope the data farming metaphor helps researchers think broadly about how their talents and interests might grow the capability in one or more of these areas, and anticipate the needs that practitioners will face in the future. We hope this metaphor resonates with practitioners, allowing them to reap immediate benefits by using a data farming approach for their next simulation study. We hope that the breadth and depth of insights that can be gleaned will help decision makers in the public and private sectors turn to simulation as a means of obtaining useful, robust, and actionable recommendations to address the complex problems they face.

Chapter 2

Randomness in Modeling

Key Concepts

- Randomness provides a powerful tool for modeling complexity.
- Randomness is an integral aspect of most complex systems, not a nuisance to be sidestepped or avoided.
- Means and variances can address some types of questions, but are completely inappropriate for others. It's often better to tackle a problem by understanding quantiles or risk.

2.1 Randomness can simplify models

Why would we want to introduce stochasticity into our simulations? In a nutshell, to create simple and useful models.

Let's assume for the sake of argument that the universe is completely deterministic—even though we, your humble authors, don't believe it—and see that we land squarely back on randomness to successfully model complex systems. With the deterministic assumption, for any scenario you choose to model there is one and only one correct answer. Without the complete state space of absolutely everything that determines that answer, your model will be wrong. For instance, if you want to predict how long it will take to fly from New York to London, you need to know the vector sums of *all* forces acting on the aircraft, which means you need the complete state (down to the atomic level) of the aircraft itself, the passengers, the atmosphere, fluctuations in the magnetic field of the earth, cosmic rays that can trigger upper atmospheric events, and so on ad nauseam. Excluding any aspect of the potential forces involved makes your answer wrong, because you'll be unable to calculate the correct deterministic number.

Please note that all of the above assumes you can infer the state information by observation. Kalman (1963) proved back in 1960's that there are deterministic linear systems which are non-observable, i.e., you could watch the input-output behavior for eternity and not be able to infer the internal state of the system.

Clearly, there's no way to measure it all, and even if there was, there's no way to maintain so much state information in any computing device we can build. And so we simplify and acknowledge that there is some degree of uncertainty in our model's predictions/solutions. We back off the concept of deterministic truth and describe results as being close to some true but unknown value with some margin of error.

Once you embrace the existence of uncertainty, it leads directly to stochastic models. One philosophical view of probability is that it is a mathematical formalism for modeling uncertainty. Rather than try to model every physical aspect of an aircraft's flight, we can characterize the likely outcomes based on what proportion of flights require less (or more) than any particular amount of time, i.e., describing the distribution of possible flight times. Having adopted distributional modeling, you can see how distributional behaviors propagate though other parts of a system—either analytically, if your system is sufficiently simple, or by generating realizations of the distributions and using replication and sampling via simulation.

Many people think that by adding more detail to a model they are increasing the model's accuracy. Prof. Bill Maxwell of Cornell University liked to use "how long does it take to cook a pizza?" to show that this can be a misconception. In his narrative a group of mechanical engineers tried to model this problem using thermodynamics to describe the dough's heat conductivity, fluid flow of sauce, and plasticity of cheese, not to mention the toppings. They added a heat-flow model for the interior of the oven, influenced by the construction materials and the geometry. This was modified by measuring heat exchange to the kitchen every time the oven door was open and shut, including roiling and fanning of the air by the movement of the door. This required a model of the kitchen temperature, which had to be modified to represent the exchange of air between the kitchen and the rest of the store, which in turn was modified by the local weather. It turns out that there's no such thing as a local weather model, so they expanded the scope to regional. They also took into account the humidity levels, which affect the heat transfer properties from air to cheese, sauce, dough. Humidity levels are affected by myriad factors such as condensation on the glasses of cold drinks and evaporation rates determined by the surface area of these drinks. They even tried modeling the level of the drinks in the cups, since cups are conic sections where the surface area can vary. Each and every one of these phenomena is legitimately a factor in the cooking process from the perspective of physics, thermodynamics, and materials-science, but the end result was a horribly complex computer model which they never got debugged, and which was based on many assumptions about the correct parameterizations. A stochastic modeler would have spent a few hours gathering data, and some basic statistics would show that all of that detail could have been replaced by using uniform random numbers between 9 and 12 minutes as the time to cook a pizza.

The point of these examples is that distributional modeling, when done properly, can often simplify your model significantly, while providing results that have more utility than a detailed physics model because the stochastic version of the model can actually be implemented.

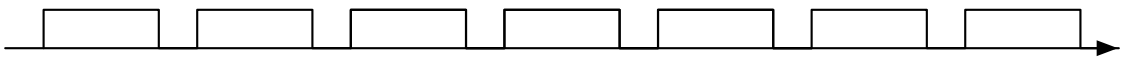
2.2 Randomness can be pivotal for good decisions

It has been common practice for decades for analysts to use means (expected values) rather than distributions in their models in an attempt to maintain simplicity. Regardless of how common it has been, this is a dangerous practice that can invalidate the analyst's conclusions. We present a few straightforward examples of how this approach can fail. Theoretical justification can be found in virtually any probability & statistics or simulation modeling textbook, and numerous articles.

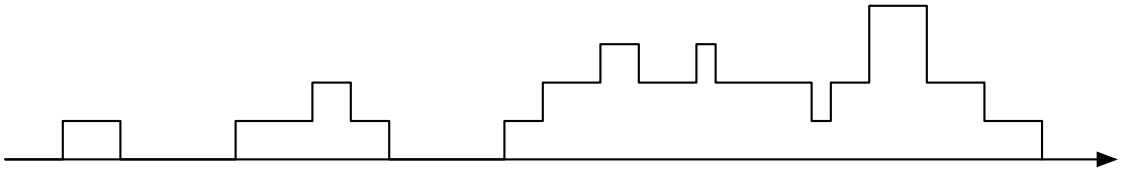
2.2.1 Randomness and inputs

Omitting randomness can radically change system behavior. One of the most striking examples of radical change when means are used to replace distributions in a model is found with simple queueing systems. A critical characteristic of all queueing systems is their traffic intensity ρ, the ratio of their arrival rate to their service rate. Both mathematics and simulation show that if the time between arrivals or the service time is random, the length of the queue and the amount of time customers spend waiting in line are both functions of ρ. Higher traffic intensities yield longer lines and larger wait times. As ρ approaches 1, the queue length and customer wait times will asymptotically grow towards ∞. At traffic intensities as high as 0.9—where the server is still idle 10% of the time—lines can spike as high as dozens or even hundreds of customers awaiting processing during busy periods. However, if you replace the distributions of interarrival times and customer service times with their means, the line length and delay in queue remain at zero for all traffic intensities up to 1, and remain at their initial value when $\rho = 1$. In other words, without randomness a queue initialized with zero customers having a traffic intensity of 1 will have no lines and no delays, while if randomness is involved the lines and delays become infinite! Less extreme examples of queue lengths appear in Figure 2.1a and Figure 2.1b for deterministic and stochastic queues, respectively.

The implications of this for combat modeling are stunning when you realize that communications, command and control decision making, and fire control are all queueing systems where the "customers" are messages, pending information requiring executive action, or ordnance waiting to be fired, respectively. Using means in lieu of distributions in a model of these processes yields model behaviors that are radically different from the real-world operations being modeled, which—by definition—invalidates the model.



(a) Deterministic interarrival and service times.



(b) Stochastic interarrival and service times.

Figure 2.1: Examples of number of customers in the system over time for a deterministic queue and a stochastic queue, both with traffic intensity $\rho < 1$.

2.2.2 Propagating randomness

A well-known result from probability is that $E[f(X)] \neq f(E[X])$ for nonlinear $f()$, but this is often overlooked by people who are unfamiliar with that field of study. The following simple counter-example demonstrates how assuming the opposite, that $E[f(X)] = f(E[X])$ is always true, can yield invalid results. If that relationship holds for any $f(x)$, then $E[X^2] = E[X]^2$. However, since $Var(X) = E[X^2] - E[X]^2$, that would make all variances zero for all random variables X.

This has significant implications for simulation models, which are implicit functions that transform the model's inputs into outputs. Virtually all simulations are nonlinear, so attempts to use the mean of an input distribution in place of random occurrences from that distribution to simplify modeling will yield different average results for the model's output.

2.2.3 Randomness and outputs

Introductory statistics courses tend to emphasize means, so it's no surprise that many analysts focus on these as a primary response measure. However, means and confidence intervals can be inappropriate or uninformative in many situations.

Suppose you need to have 7 vehicles out of a pool of 10 available on a given day to be mission-capable. Policy A has 7 vehicles available on half the days, and 9 vehicles available on the remaining half, so the expected number of vehicles on a randomly chosen day is $\mu_A = 8$. Policy B has 6 vehicles available on half the days, and 10 vehicles on the remaining half, which also has an expected value of $\mu_B = 8$. If you judge based solely on expected value, both policies would be deemed acceptable. However, notice that the command is mission-capable 100% of the time under Policy A, but only 50% of the time with Policy B. It should be clear that choosing a policy based solely on the expected number of vehicles can result in a flawed decision. Also note that the mean is never observed as an actual outcome under

either policy. Finally, be careful to not misinterpret the meaning of a confidence interval for μ_A or μ_B—these CIs give bounds on the precision with which we know the means. They can be made arbitrarily narrow by increasing the sample size. They do not give us direct information about the range of possible outcomes. That would require a prediction interval if the normality assumption is valid, or more generally the use of percentiles.

A second example illustrates how using distributions can result in better mission planning. Suppose a critical rescue mission requires 4 or more helicopters to succeed, and we know from very solid historical data that the reliability of an individual helicopter is 0.8, or 80%. A mission planner who bases the decision on expected values would say that we should plan our mission with 5 helicopters. The expected number of operational helicopters would then be $5 \times 0.8 = 4$, so in expectation we have the right number of helicopters to accomplish the mission. However, proper risk modeling yields a very different view. If we assume that helicopter failures are independent, then the number of available helicopters has a binomial$(n, p = 0.8)$ distribution, where n is the fleet size chosen for the mission. Table 2.1 shows how the probability of having fewer than 4 helicopters changes for various values of n. You can see that if the decision is based on the expected value criterion, the chances are greater than 1 in 4 that the mission will fail, as happened in the 1980 Iranian hostage rescue mission. Increasing the number of helicopters to 6 reduces the risk of failure to less than 1 in 10, and using 7 would reduce the risk to less than 1 in 30.

Table 2.1: Probability of failure by number of helicopters, no correlation

Initial helos (n)	5	6	7
P(Available helos < 4)	0.263	0.099	0.033

Be aware that even this distributional model will underestimate the risk of mission failure if helicopter failures are positively correlated. This could occur, for example, when the failures are due to environmental conditions such as tainted fuel or a sandstorm. In the extreme, where failures are perfectly positively correlated, then the P(Available helos < 4) is 0.20 for any $n \geq 4$.

A third example illustrates how focusing on means can completely mislead the analyst's perception of risk. When a defensive emplacement is tasked with shooting down an incoming missile, there is a time limit imposed by the transit time of the missile. Your window of opportunity is the time between acquisition of orders and successful targeting, and the time when the missile will impact if not shot down. Given a modern missile's ability to do lethal damage, the average time to shoot down a missile is not the appropriate measure of performance. Instead, you should be concerned about whether the time required to kill a missile *ever* exceeds the remaining transit time. This is seen as the "probability of a hit," which is the area to the right of the "Time of Impact" in Figure 2.2.

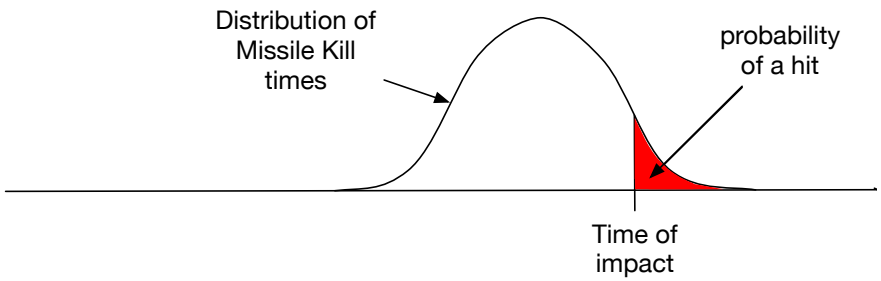


Figure 2.2: The relevant measure is probability of a hit, not the mean of Missile Kill Time.

2.3 Summary

Randomness is an essential component of good modeling. Properly used, it can help simplify your models. Ignoring randomness can drastically impact the accuracy and validity of those models. However, care must be taken to correctly assess the model's outputs when randomness is involved. Measures other than means, such as quantiles or risk, can provide more pertinent information to decision makers about important problems, yielding better decisions.

We'll finish this chapter with a cautionary note. Most programming languages, statistical packages, or spreadsheets provide libraries or modules for generating randomness. Cryptographic-quality random number generators exist, but they are computationally expensive and their output is intentionally not reproducible, which can make debugging extremely challenging. For stochastic modeling most people use "pseudo-random number generators" (PRNGs). Ideally PRNGs will generate reproducible sequences that are statistically indistinguishable from true randomness. Unfortunately, not all PRNGs meet this ideal. Regardless of the context in which you're working, it's worth double checking the documentation to make sure your modeling platform is using a modern well-tested PRNG. Spreadsheets are often among the worst offenders in this regard. Our recommendation is to not use your spreadsheet's built-in PRNGs or random variate generation for critical decision making. If you absolutely must use a spreadsheet, consider downloading plug-ins to provide well-tested algorithms.

Chapter 3

Design of Experiments

Key Concepts

- Design of experiments is a powerful way to identify cause-and-effect relationships between factors (varied at deliberately chosen levels) and responses.
- More levels are better for understanding how quantitative factors influence the performance.
- More levels improve lack-of-fit detection when metamodeling.
- The curse of dimensionality means that more levels require more design points in a brute-force approach, and quickly makes naive designs untenable as the number of factors grows.
- New designs break this curse of dimensionality.

In simulation, design of experiments (DOE) is a powerful way to identify cause-and-effect relationships between deliberately varied factors (i.e., the simulation's inputs) and the corresponding changes in responses. A well-designed experiment will generate the output needed for metamodeling in an efficient and informative way.

This chapter is organized as follows: Section 3.1 describes several simulation models that we will refer to throughout this chapter. Reference models for other chapters will be introduced when appropriate. Section 3.2 provides notation and definitions, and highlights some of the pitfalls that analysis is prone to in the absence of well-designed experiments. Section 3.3 introduces some classical designs that have been extensively and successfully used in physical experimentation, but which we do NOT recommend for large-scale data farming studies. Our motives for including them are that people exploring the field of DOE should have some grasp of its roots; and their relative simplicity is useful for motivating important principles of DOE. Subsection 3.3.7 presents extensions of the classic designs that were developed specifically

with data farming in mind. Section 3.4 introduces modern high-dimensional space-filling designs. These are designs that we strongly recommend and use most often in our own data farming studies. Section 3.5 describes some run control approaches and briefly discusses how cluster or cloud computing can be leveraged. Finally, Section 3.6 presents several other considerations you should take into account for large-scale data farming studies.

We have chosen to separate the discussion of experiment designs in this chapter from the analysis of the response data in subsequent chapters. We cannot completely separate design and analysis since the choice of design impacts the types of analyses that can be done, and the types of analyses we wish to conduct impacts the way we must grow the data. Consequently, we assume the reader already has a basic familiarity with multiple regression metamodeling and its corresponding terminology, such as main effects, quadratic effects, two-way interactions, and degrees of freedom (d.f.). If not, the reader may wish to read Sections 4.1 and 4.3.1 before continuing with this chapter.

3.1 Reference Simulations

available code

`capture_the_flag.rb`
: terminating simulation for notional capture-the-flag game

`gummies.rb`
: static simulation for gummy bears in space

`epidemic.rb`
: stochastic implementation of a basic epidemiological model

`fleet_availability.rb`
: non-terminating simulation for fleet operational availability

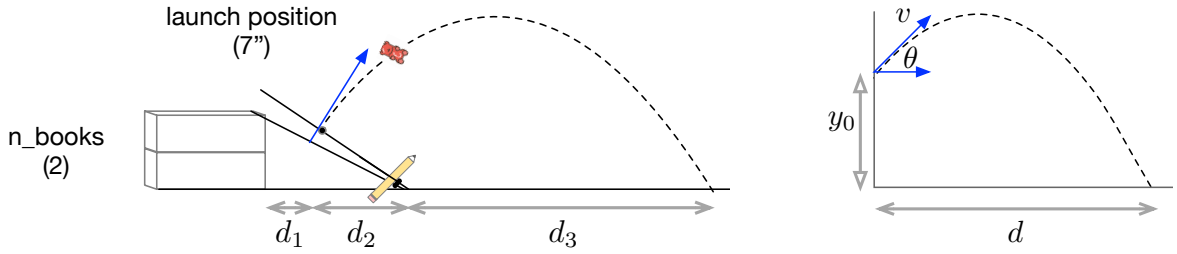
3.1.1 Capture the Flag

Consider an agent-based simulation model of a game of capture the flag, where two teams are trying to find and take the opposing team's flag while simultaneously protecting their own flag. This is an example of a terminating simulation, where the simulation ends at a clearly defined (but random) point in time when one of the teams wins the game by fulfilling the pre-defined victory criteria. This is a notional example, i.e., we do not have actual agent-based code available to distribute. However, over the years we have developed a surrogate that produces results in accordance with the graphs and discussion we show in this book, as well as in earlier tutorials and papers.

3.1.2 Gummy Bears in Virtual Space

This example is based on "Gummy bears in space" described in a wonderful book on Activity-based Statistics by Scheaffer et al. (1996). In the physical experiment described in this book, student teams can construct a simple catapult using two 12-inch (or 30-cm) rulers, a rubber band, and a pencil. The catapult is created by holding the rulers together and wrapping the rubber band tightly at the one-inch (2.5cm) mark, and inserting the pencil as a spacer between the rulers. The base of the catapult (0 inches/cm on the ruler) is placed on a table, and the catapult is propped up on books. Gummy bears are just sticky enough to lay on the ruler without sliding down, and can be launched by depressing and then releasing the upper of the two rulers.

We can do a similar simulation experiment with virtual gummies in virtual space, using the `gummies.rb` program. This is a static simulation, defined in Smith and Sturrock (2024) as "one in which the passage of time plays no active or meaningful role in the model's operation and execution." Figure 3.1a shows the basic setup. Rather than using a rubber band and pencil to give the catapult its springiness, we'll simply generate the velocity at which the gummy leaves the ruler.



(a) Gummy bears in space

(b) Range of a projectile.

Figure 3.1: Logic underlying the virtual gummies in space simulation

There is a physics-based formula for the range of a projectile. The graph in Figure 3.1b shows that a projectile launched from height y_0 with velocity v at angle θ will travel a horizontal distance d. The formula is;

$$d = \frac{v^2}{2g}\left(1 + \sqrt{\frac{2gy_0}{v^2 \sin^2\theta}}\right)\sin 2\theta \tag{3.1}$$

where g, the coefficient of gravity, is 9.81 meters/second^2 (or 32 feet/second^2).

3.1.3 Epidemic modeling

This is a stochastic epidemic simulation, based on the *Susceptible* → *Exposed* → *Infectious* → *Removed* (SEIR) compartmental model in epidemiology. At time zero, a specified number of individuals in a fixed population are initially infected. Each infected individual spends some time in an exposed but non-infectious state (E), then switches to the infectious state (I) and becomes capable of infecting other susceptible individuals (S). After spending some time in the infectious state, the individual transitions to the removed state (R). In this model, that means that the individual either has died or has recovered and is immune to future infections. The simulation allows the user to specify an intervention time and efficacy, so the rate of disease transmission decreases after the intervention time. This terminating simulation stops once there are no longer any infected (i.e., exposed or infectious) individuals.

3.1.4 Fleet Availability

This is a non-terminating simulation. A distribution center has a fleet of vehicles. Over time, vehicles become due for scheduled maintenance and are serviced, or break down and require repairs. Planners base daily operations on the availability of vehicles at the beginning of each day. The proportion of the initial fleet available is called the operational availability (people with U.S. military experience may recognize this as "Ao.") The fleet might be comprised of logistics delivery assets such as trucks, ships, or drones. Alternatively, it could correspond to a set of machines in a job shop, where maintenance and breakdowns can reduce availability at the beginning of each day.

3.2 Notation and Definitions

Design of Experiments (DOE) is a subfield of statistics focused on efficient data generation plans that provide informative results. Initially, DOE was developed for physical experiments, often in agricultural, manufacturing, or clinical settings. However, as computing power became available, the use of designed experiments expanded to include deterministic computational models as well as stochastic simulation models. The basic DOE concepts are the same in all of these settings. However, with physical experimentation our ability to generate observations is much more constrained, so using designs intended for physical experiments on your stochastic simulations or deterministic computer models can substantially limit the breadth and depth of insights you can gain.

In general, a *design* is a matrix where the columns correspond to the factors, the entries correspond to *levels* to be assigned to these factors, and each row represents a particular combination of factor levels known as a *design point*. If the entries in the rows correspond to the actual settings we'll be using for the experiment, we will call these the *natural levels*. Designs are often constructed in a problem-independent, normalized form using *coded levels*, and then converted to natural levels. For example, consider the gummy bear model described in Section 3.1.2. One possible design (in natural levels) is shown in Table 3.1.

Table 3.1: Gummy bear design (natural levels)

Design Point	Launch Position (inches)	No. of Books
1	4	1
2	8	1
3	4	2
4	8	2
5	4	3
6	8	3

Using coded levels is a convenient way to allow us to reuse the same base experiment design for any study involving the same number of factors studied at the same numbers of levels. Different coding schemes are possible, but a convenient one for quantitative data is to denote the lowest and highest levels of interest by -1 and $+1$, respectively. The base design for the gummy bear experiment is shown, using this coding scheme, in Table 3.2. Finally, if there are only two or three possible levels for each factor, we can cut down on the amount of ink we need to print the design by just reporting the signs, as in Table 3.3.

Table 3.2: Gummy bear design (coded numerical levels)

Design Point	Launch Position	Books
1	-1	-1
2	1	-1
3	-1	0
4	1	0
5	-1	1
6	1	1

Table 3.3: Gummy bear design (coded $-/0/+$ levels)

Design Point	Launch Position	Books
1	$-$	$-$
2	$+$	$-$
3	$-$	0
4	$+$	0
5	$-$	$+$
6	$+$	$+$

Quantitative factors are numeric-valued. They are classified as *continuous* when the underlying range of potential experimentation is continuous, and as *discrete* when potential experiments are limited to a countable set of values. For example, consider the `gummies.rb` simulation. Launch position is a continuous-valued factor since we can launch gummies from

any position on the ruler. Conversely, the number of books is a discrete factor because only integer-valued levels are allowed.

A third type of factor is a *qualitative factor*, also called a *categorical* or *nominal* factor. An example would be the color of the gummy bear being launched. Whether a factor is quantitative or qualitative, if it has only two potential levels (such as yes/no, on/off, or 0/1) it may be called a binary factor. Different classes of designs may be needed for different types of factors. In general, continuous-valued factors are the easiest to deal with.

Note that these classifications are sometimes determined by the simulation implementation, rather than the intrinsic nature of the system. For example, in `gummies.rb` the launch height is a discrete factor, with levels entered as an integer number of books. A different implementation of the gummies simulation might model "launch height" as a continuous factor with levels entered as inches or cm in decimal form. Alternatively, continuous ranges can sometimes lead to categorical factors if only a few potential levels are of practical interest. For example, in `fleet_availability.rb` the scheduled maintenance is a discrete quantitative factor specified in terms of days between scheduled maintenance. A different implementation of the fleet availability simulation might model "scheduled maintenance" as a qualitative factor with levels of none, weekly, monthly, or annually.

Factor interactions describe a situation where the effect a given factor has on the response varies depending on the level(s) of one or more other factors. For example, your comfort level depends on the air temperature but the temperature effect is modified by wind speed—breezes reduce your discomfort at higher temperatures, while they chill you faster at lower temperatures.

If we repeat the whole matrix, this is called a *replication* of the design. Let n_d be the number of design points, and n_r be the number of replications. Then the total number of *experimental units* is $n_{tot} = n_d n_r$. In simulation, the scope of what is meant by an experimental unit depends on the type of simulation model being studied, as shown by the following examples:

- Static: A single observation, such as the distance measurement for a single gummy launch from `gummies.rb`;
- Terminating: An aggregate, average, or final value from a single run of a terminating simulation, such as the average time customers are in a supermarket during a particular day, or the duration of an epidemic from a single run of `epidemic.rb`; or
- Non-terminating: An aggregate or average value from a run or batch of a non-terminating simulation after removing any initial transient behavior, such as the 10th or 90th percentile of the daily start-of-day availability in the `fleet_availability.rb` simulation. Other examples could be the batch mean or batch standard deviation of the time patients spend in the emergency room in a hospital simulation.

Randomization, Replication, and Control

Three important concepts in DOE are *randomization*, *replication*, and *control*. However, there are differences in how these concepts influence the experiment designs used in different settings.

In a physical experiment, you may need to randomize the order of the design points. Randomness helps guard against any unidentified sources of bias in the way you collect data. There are several alternatives, including:

- Completely randomizing over all n_{tot} rows;
- Completely randomizing within each replication (e.g., within each set of n_d rows); or
- Randomizing the order of the design points, and taking all n_r replications on one design point before moving on to the next.

Replication is important because the amount of data you gather determines how powerful your tests of significance are (i.e., how small your p-values can get), and how narrow your confidence intervals will be. Since physical experiments are often costly to conduct, methods for keeping n_{tot} small may be of interest. Variability in a response is often considered to be a nuisance, so experimenters attempt to remove as many potential sources of uncontrolled variation as possible from the experiment setting.

Control also can be difficult in physical considerations, which contributes to the historical mindset that experiments should test a limited number of hypotheses involving a small number of factors. Consequently, this can mean there are many uncontrolled sources of variation that affect the responses in unknown or unpredictable ways. For example, in the physical gummy bear experiment of Table 3.1, the only two factors are the number of books and the launch position on the ruler. Uncontrolled sources of variation could arise if more than one person launches the gummies, if different colors of gummies have slightly different weights or shapes, if varying environmental conditions such as humidity or temperature affects the stickiness of the gummies, if a nearby window is open for some of the launches, or if different people record the measurements differently.

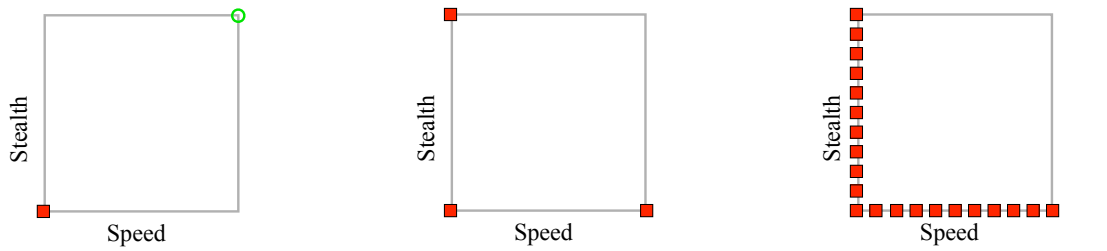
In deterministic computer experiments, randomization and replication are non-issues. Control is easy, in the sense that the analyst can control everything in the deterministic model or equation.

In stochastic simulation experiments, randomness is an integral part of the system. Care is taken in the choice of random number generators, and greater control can be applied by using common random numbers (CRN) instead of independent runs. Variability in the response is typically interesting in and of itself, and should not be considered just a nuisance. However, unlike in physical experiments, there is no need to randomize the order of runs to reduce potential bias due to unknown sources of variation. We are in charge of everything in a programmatic simulation environment, so control is easy relative to physical experiments.

Common pitfalls

An overarching aspect of control is specifying the experiment design, i.e., choosing the factors, factor levels, and design points to be studied. There are many good designs, but bad designs also exist—and should be identified as such and avoided.

Two common types of simulation studies are, in fact, ill-designed experiments. The first can occur if several people each suggest a few "interesting" scenarios—particular combinations of factor settings—so a haphazard handful of design points end up being explored where the levels of many factors are changed simultaneously. Consider the capture-the-flag simulation model of Section 3.1.1. Suppose that only two design points are used, corresponding to different settings for speed (X_1) and stealth (X_2), with the results shown in Figure 3.2a. We depict "good" average outcomes as green circles, and "bad" average outcomes as red squares. One stakeholder might claim these results show that high stealth is of primary importance, another that speed is the key to success, and a third that they are equally important. There is *no way* to resolve these differences of opinion without collecting data from additional design points. In statistical terms, the effects of stealth and speed are said to be *confounded*. In practice, simulation models can easily have dozens or hundreds of potential factors. A handful of haphazardly chosen scenarios, or a trial-and-error approach, can use up a great deal of time and effort without addressing the fundamental goals of the study.



(a) Confounded effects

(b) One-factor-at-a-time variation with 2 levels

(c) One-factor-at-a-time variation with many levels

Figure 3.2: Bad designs for capture the flag.

Another type of problematic study occurs when people start with a "baseline" scenario and vary one factor at a time. Revisiting the capture-the-flag example, suppose the baseline corresponds to low stealth and low speed. Varying each factor, in turn, to its high level yields the results of Figures 3.2b and 3.2c, It appears that *neither* factor is important, so someone using the simulation results to decide how to train and equip their team would not know how (or if) to proceed. However, if all four combinations of speed and stealth (low/low, low/high, high/low, and high/high) are sampled, as in Figure 3.3, it is clear that success requires both high speed and high stealth. This means that the factors interact—and if there are factor interactions, one-factor-at-a-time sampling will never uncover them! In the multitude of systems we've studied, our experience is that interactions are pervasive and important. When a subject-matter expert is asked about the effect of a particular factor, a

very common (and correct!) response is "it depends. . . ," which is an implicit way of pointing out the existence of interactions.

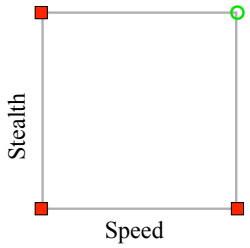


Figure 3.3: 2×2 grid sampling for Capture the Flag

The pitfalls of using a poor design seem obvious on this toy problem, but the same mistakes are made far too often in larger studies of more complex models. When only a handful of scenarios will be explored, there is a potential hazard of confounding not just two, but dozens or even hundreds of factors. Similarly, many analysts change one factor at a time from their baseline scenario and consider this a way of performing sensitivity analysis. In doing so, they fail to understand that this approach implicitly assumes that there are no interaction effects. Using a restricted range for the factors may appear to fix this in terms of the analysis, but means your study is of limited use since it will not reveal the presence of interactions and thus will be invalid outside the limited range explored.

3.3 Classical Designs

The designs in this section are provided to highlight some of the guiding concepts in experiment design. However, we want to reiterate that the designs in Section 3.3.7 and Section 3.4 are more suitable for large-scale data farming studies.

3.3.1 Factorials or Gridded Designs

helpful datafarming gem scripts

`cross.rb` *`filenames...`*
create a combinatorial design by crossing all combinations of any number of individual smaller designs in named files

Factorials (or gridded designs) are perhaps the easiest to discuss: they examine all possible combinations of the factor levels for each of the X_is. A factorial design is described as an m^k full factorial when k factors are investigated, each at m levels. This notation also tells us the total number of design points is m^k. If different factors have different numbers of levels, we can still determine the number of design points using a product form. For example, if

k_1 of the factors have m_1 levels and k_2 of the factors have m_2 levels, the number of design points is $m_1^{k_1} \times m_2^{k_2}$. This notation can be generalized to arbitrary numbers of groups where each group shares a common number of levels per factor. As a concrete illustration, if one study does the gummy bear experiment with 3 launch angles and 2 launch positions, this is a $3^1 \times 2^1$ factorial experiment with 6 design points. If another study uses 3 launch angles and 3 launch positions, that's a 3^2 factorial experiment with 9 design points.

More generally, consider designs two A and B:

$$A = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{pmatrix}, \qquad B = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}, \tag{3.2}$$

where a_i for $i = 1, \ldots, m$ and b_j for $j = 1, \ldots, n$ can each be comprised of multiple columns. Crossing these two designs produces a new design:

$$A \otimes B = C = \begin{pmatrix} a_1 & b_1 \\ a_1 & b_2 \\ & \vdots \\ a_1 & b_n \\ a_2 & b_1 \\ a_2 & b_2 \\ & \vdots \\ a_2 & b_n \\ & \vdots \\ a_m & b_1 \\ a_m & b_2 \\ & \vdots \\ a_m & b_n \end{pmatrix} \tag{3.3}$$

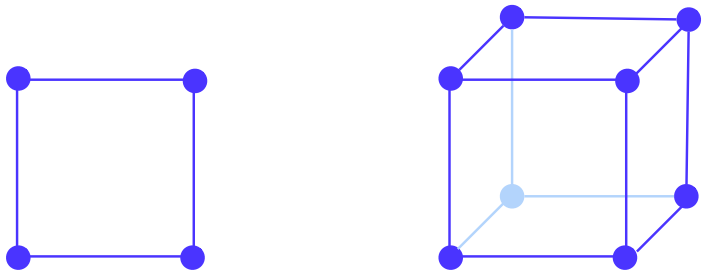
where $\otimes$ denotes the cross operation. The resulting design has $m \times n$ design points (rows). This can be generalized to multiple designs using a recursive approach, where the first two designs are crossed, the resulting design is crossed with the third, the result of that is crossed with the fourth, and so on. The resulting design has the product of all the contributing designs' design sizes as its number of design points.

The most commonly-used factorial design is a 2^k because it requires only two levels (low and high) for each factor. Note that this design restricts analysis to linear models, since we cannot estimate any quadratic effects if we only observe factors at two levels. 2^k designs are very easy to construct. Simply figure out $n_{dp} = 2^k$ and populate a k by 2^k matrix as follows. The first column alternates + and −, the second column alternates in groups of 2, the third column alternates in groups of 4, and so forth. Table 3.4 shows a 2^3 factorial design.

Table 3.4: 2^3 factorial (coded units)

Design Point	X_1	X_2	X_3
1	−	−	−
2	+	−	−
3	−	+	−
4	+	+	−
5	−	−	+
6	+	−	+
7	−	+	+
8	+	+	+

If you are using a spreadsheet, you can easily move from a design for k factors to a design for $k+1$ factors by copying the 2^k design, pasting it below to obtain a $2^k \times k$ matrix, and then adding a column for factor $k+1$ with the first 2^k values set to -1 and the second set of 2^k values set to $+1$. Conceptually, 2^k factorial designs sample at the corners of a hypercube defined by the factors' low and high settings. Figure 3.4 shows examples for 2^2 and 2^3 designs. Envisioning a 2^4 or larger design is left to the hyperimaginative reader.



(a) 2^2 design **(b)** 2^3 design

Figure 3.4: Graphical representation of small factorial designs.

Now, what happens when we go to analyze the results? First, let's see how we'd construct columns for the interaction terms if we had to do this in Excel. (Fortunately, reputable statistics software handles interactions nicely behind the scenes). We could multiply the columns without subtracting off the column means since everything is already centered around zero. You can easily verify that the interaction columns in Table 3.5 are found simply by multiplying the corresponding columns for the main effects.

Table 3.5: Terms for a 2^3 factorial

Design Point	Term						
	X_1	X_2	X_3	X_1X_2	X_1X_3	X_2X_3	$X_1X_2X_3$
1	−	−	−	+	+	+	−
2	+	−	−	−	−	+	+
3	−	+	−	−	+	−	+
4	+	+	−	+	−	−	−
5	−	−	+	+	−	−	+
6	+	−	+	−	+	−	−
7	−	+	+	−	−	+	−
8	+	+	+	+	+	+	+

Recall that a linear regression model for predicting response Y as a function of three factors x_1, x_2, and x_3 would be

$$\begin{aligned} E[Y] = \beta_0 &+ \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 \\ &+ \beta_{1,2} x_1 x_2 + \beta_{1,3} x_1 x_3 + \beta_{2,3} x_2 x_3 \\ &+ \beta_{1,2,3} x_1 x_2 x_3. \end{aligned} \tag{3.4}$$

Note the correspondence between terms in equation 3.4 and the terms in Table 3.5. We see that there are seven terms corresponding to the factor's effects on the response—three main effects, three two-way interactions, and one three-way interaction—that could potentially be estimated from a 2^3 factorial experiment. Of course we would also want to estimate the intercept β_0, which is the overall mean, so there are 8 coefficients to be estimated from 8 data points. That could be solved algebraically if this were a deterministic system, but for a stochastic system that's not enough data. We will always need at least 1 d.f. for estimating error, and preferably a few more.

If we increase the number of factors k, we find a similar relationship. In general, there will be k main effects, $\binom{k}{2}$ two-way interactions,..., $\binom{k}{k-1}$ different $(k-1)$-way interactions, and one k-way interaction (i.e., $\binom{k}{i}$ i-way interactions for $i \in 1, ..., k$). If we add all these up, we get a total of $2^k - 1$ terms plus the intercept. Once again, we won't be able to estimate everything because there won't be any d.f. left over for error. Table 3.6 shows the matrix for a 2^4 factorial. To save a little room on the headings, we've left out the Xs and just given the factor numbers. There are 16 design points and 15 effects (4 main effects, 6 two-way interactions, 4 three-way interactions, and 1 four-way interaction).

There are two basic options to deal with this situation:

- **Replicate** the design to get more d.f. for error. This will also help you detect smaller effects by reducing the standard errors of their estimates. For example, adding a single replication to the base design would reduce the standard error of means (or slopes) by roughly $1/\sqrt{2}$. More replications would increase our discriminating power even further.

Table 3.6: Terms for a 2^4 factorial

Design	Term														
Point	1	2	3	4	1,2	1,3	1,4	2,3	2,4	3,4	1,2,3	1,2,4	1,3,4	2,3,4	1,2,3,4
1	−	−	−	−	+	+	+	+	+	+	−	−	−	−	+
2	+	−	−	−	−	−	−	+	+	+	+	+	+	−	−
3	−	+	−	−	−	+	+	−	−	+	+	+	−	+	−
4	+	+	−	−	+	−	−	−	−	+	−	−	+	+	+
5	−	−	+	−	+	−	+	−	+	−	+	−	+	+	−
6	+	−	+	−	−	+	−	−	+	−	−	+	−	+	+
7	−	+	+	−	−	−	+	+	−	−	−	+	+	−	+
8	+	+	+	−	+	+	−	+	−	−	+	−	−	−	−
9	−	−	−	+	+	+	−	+	−	−	−	+	+	+	−
10	+	−	−	+	−	−	+	+	−	−	+	−	−	+	+
11	−	+	−	+	−	+	−	−	+	−	+	−	+	−	+
12	+	+	−	+	+	−	+	−	+	−	−	+	−	−	−
13	−	−	+	+	+	−	−	−	−	+	+	+	−	−	+
14	+	−	+	+	−	+	+	−	−	+	−	−	+	−	−
15	−	+	+	+	−	−	−	+	+	+	−	−	−	+	−
16	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+

- **Make simplifying assumptions.** The most common type of assumption is to assume that higher-order interactions aren't present. For example, if we assume the three-way interaction is negligible in the system being studied, then we can exclude that column in the 2^3 factorial (Table 3.5) from our analysis. We would then have 1 d.f. for estimating error and we could fit a metamodel involving 3 main and $\binom{3}{2} = 6$ two-way interaction effects once we ran the experiment. Similarly, if we could assume there is no four-way interaction in the system studied with Table 3.6, then we gain 1 d.f. for error by excluding the corresponding term from our model. If we further assume there are no three-way interactions as well, we'd have 5 d.f. for error.

Making simplifying assumptions sounds like a potentially dangerous thing to do, but it's often not bad. Over the years, statisticians conducting field experiments have found that often, if there are interactions present, the main effects will also show up (unless you "just happen" to set the low and high levels so everything cancels out). There's also a "rule-of-thumb" from these field experiments that if two-way interactions exist, their slopes are only about 1/3 the size of the main-effect slopes; if three-way interactions exist, their slopes are only about 1/3 the size of the two-way slopes; etc. Finally, there's the pragmatic question of how would you explain a four-way interaction in human comprehensible terms.

One caveat: most of these historical field experiments involved only a few (1–5) factors because of the difficulty in controlling them and the high costs of physical experimentation, both of which contribute to tight constraints on the number of design points. Statisticians knew ahead of time they would need to make compromises, and tried to set up the experiments so they felt comfortable making the assumptions outlined above. For example, since smooth curves can be well-approximated by piecewise linear segments over small ranges, the

guidance was to vary factors over sufficiently small ranges that nonlinearity was thought not to be a concern. Our experience in large-scale simulation analysis is different. We may expect stronger interaction effects in a logistics simulation scenario than when growing fields of potatoes.

Overall, factorial designs have several nice properties if you're running a small experiment. They are easy to construct by hand, which was an important consideration in the days before modern computers. They are simple to explain—even people who don't know anything about DOE are comfortable with the idea that you're looking at "all possible combinations" of the factor levels of interest. They let us examine more than one factor at a time, so they can be used to identify important interaction effects. They are also *orthogonal* designs—the pairwise correlation between any two columns (factors) is equal to zero. This simplifies the analysis of the output (Ys) we get from running our experiment, because estimates of the factors' effects (their $\hat{\beta}_i$s) and their contribution to the explanatory power (R^2) of the regression metamodel will not depend on what other explanatory terms are present or absent in the regression metamodel.

3.3.2 m^k Factorials (Finer Grids)

helpful datafarming gem scripts

`cross.rb` *`filenames...`*
: create a combinatorial design by crossing all combinations of any number of individual smaller designs in named files

`cat.rb` *`filenames...`*
: Unix `cat` program work-alike for Windows—for viewing or creating text files, or concatenating multiple files, based on stdio

`csv2blank.rb` *`filenames...`*
: convert CSV files to whitespace-delimited files

Examining each of your factors at only two levels (the low and high values of interest) turns out to be very efficient. Unfortunately, it means you have no idea what's happening in between. There are many situations where you'd like a better idea of what's happening for intermediate values. An obvious solution is to use finer grids.

A design with k factors, each at 3 levels, is called a 3^k factorial. In general, a design having m levels for each of its k factors is called an m^k factorial. This nomenclature directly describes the number of design points. For $m = 3$, the convention is to use -1, 0, and 1 (or $-$, 0, and $+$) for the coded levels. As we move to larger values of m, it's common to specify the design in the natural units of the problem, but we retain the m^k nomenclature to describe the design.

Consider the capture-the-flag example once more. Figure 3.5 shows the (notional) results of two experiments: a 2^2 factorial on the left and an 11^2 factorial on the right. In both subgraphs the green circles—including the upper right-hand corner—represent good results, the yellow triangles in the middle represent mixed results, and the red squares on the left-hand side and bottom represent poor results. For the 2^2 factorial, all that can be said is that when speed and stealth are both high, our team is successful. Much more information is conveyed by the 11^2 factorial. We see that if the agent can achieve a minimal level of stealth, then speed is more important.

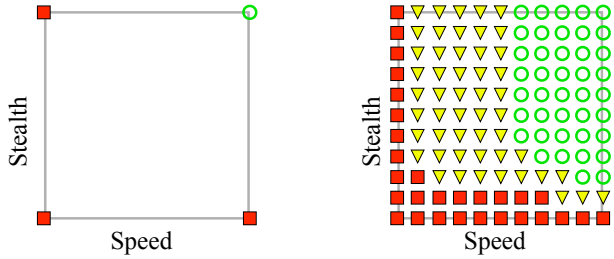


Figure 3.5: 2^2 and 11^2 Factorial Experiments for Capture the Flag

The larger the value of m for an m^k factorial design, the better its space-filling properties. A scatterplot matrix of the design points shows projections of the full design onto each pair of factors. Consider the graph in Figure 3.6 for a 5^4 factorial. The graph contains cells of subplots of the design points for pairs of factors at a time. For instance, the third cell over in the top row plots the (X_3, X_1) factor combinations, and the third cell down in the left column is just its transpose, plotting the pairs (X_1, X_3), and thus carries the same information. Each subplot has four points in the corners, three additional points along each edge, and nine points in the interior. The corresponding subplots for a 2^4 factorial would each reveal only four points, one at each corner. The bad news is that while the finer grid provides more information, that comes at a cost of 625 design points instead of 16.

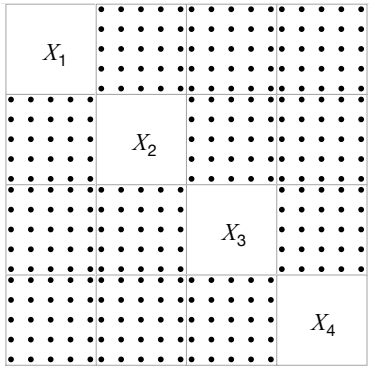


Figure 3.6: Scatterplot Matrix for a 5^4 Factorial Design

3.3.3 The Curse of Dimensionality

Even 2^k becomes large very quickly. Finer-grid designs are even worse in terms of data requirements. For example, consider the experiment sizes in Table 3.7. The entry "ginormous" denotes a design that has far too many dps to even consider.

Table 3.7: Design points required for full factorial designs (single replications)

# factors	2^k factorial	5^k factorial	10^k factorial
1	2	5	10
2	$2^2 = 4$	$5^2 = 25$	$10^2 = 100$
3	$2^3 = 8$	$5^3 = 125$	$10^3 = 1,000$
5	32	3,125	100K
10	1,024	9,765,625	10 billion
20	1,048,576	95 trillion	ginormous!

When we look at this and think about how many multi-way interactions we *could* fit but may not believe are important (relative to main effects and two-way or possibly three-way interactions), this seems like a lot of wasted effort. It means we need a *smarter, more efficient* type of experiment design if we have a large number of factors.

3.3.4 2^{k-p} Fractional Factorials

helpful datafarming gem scripts

`generate_design.rb -d resv` *`options/range-specs`*
generate 2-level resolution V fractional factorial (R5FF) designs with factor scaling

If we are willing to assume that some high-order interactions aren't important, then we can reduce the number of design points that are required for a factorial experiment. The magnitude of the reduction is dramatic as the number of factors increases. We will illustrate this for a 2^3 factorial, but the concepts applied in this example generalize broadly. Consider the design of Table 3.4, and suppose that we are willing to assume that there are NO interactions. It's easy to confirm that the rightmost column of the design in Figure 3.7a contains the outcomes corresponding to a 3-way interaction, but under our assumption of no interactions we can call this column X_4 and investigate 4 factors in $2^3 = 8$ design points—half the size of the full factorial design for 4 factors! This is denoted as a 2^{4-1} fractional factorial, and shown graphically in Figure 3.7b. We would be able to estimate (or test) 4 different factors in 8 design points. Each of the subplots in the scatterplot matrix of Figure 3.7b shows only 4 points instead of 8 because some are projected on top of each other.

Better still, as long as we're assuming no interactions, we could squeeze a few more factors into the study. We could substitute new factors for all of the interaction terms in the full factorial design in Table 3.5, yielding Table 3.8.

Table 3.8: 2^{7-4} resolution III fractional factorial (R3FF) design, in coded units.

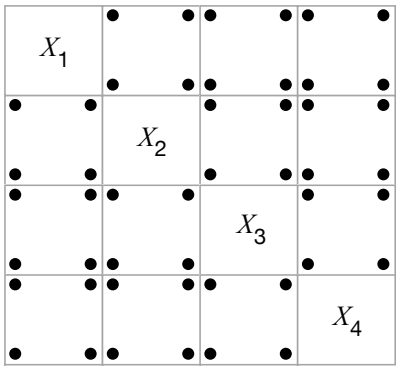
Design Point	Factor (was)	X_1	X_2	X_3	X_4 (X_1X_2)	X_5 (X_1X_3)	X_6 (X_2X_3)	X_7 $(X_1X_2X_3)$
1		−	−	−	+	+	+	−
2		+	−	−	−	−	+	+
3		−	+	−	−	+	−	+
4		+	+	−	+	−	−	−
5		−	−	+	+	−	−	+
6		+	−	+	−	+	−	−
7		−	+	+	−	−	+	−
8		+	+	+	+	+	+	+

The resulting design is called a 2^{7-4} fractional factorial, since the base design varies 7 factors in only $2^{7-4} = 8$ design points instead of $2^7 = 128$ design points. The design is said to be *saturated* since we couldn't squeeze anything else into it, and note that we have no degrees of freedom left for estimating error. If we take 2 replications of this $n_d = 8$ experiment, we can examine 7 factors in as few as 16 simulation runs with 8 d.f. for error estimation. Alternatively, if we only have 6 factors to study we can fit a main-effects metamodel to the $n_d = 8$ experiment results where one d.f. is used for each factor and an intercept term, leaving a single d.f. for estimating error with one replication or 9 d.f. if we make two replications.

More generally, any design involving k factors that is written as 2^{k-p} for some integer p is a fractional factorial that uses a $2^{-p} = 1/2^p$ fraction of the dps of the corresponding 2^k full

Design Point	Factor (was)	X_1	X_2	X_3	X_4 $(X_1X_2X_3)$
1		−	−	−	−
2		+	−	−	+
3		−	+	−	+
4		+	+	−	−
5		−	−	+	+
6		+	−	+	−
7		−	+	+	−
8		+	+	+	+

(a) Design in coded units.



(b) Scatterplot matrix.

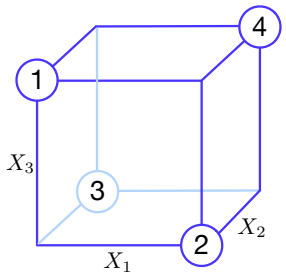
Figure 3.7: 2^{4-1} resolution III fractional factorial (R3FF).

factorial. Depending on how they were constructed, each fractional factorial has a property referred to as its *resolution*, which is often described using Roman numerals. We often use shorter notation such as R3, R5, or R7 for resolutions III, V, and VII, respectively. An R3 design permits the estimation of main effects (a first order metamodel). When using an R3 design, the analyst must assume that the main effects dominate in the simulated system, because the '3' indicates that main effects are confounded with two-way interactions ($1+2=3$). An R5 design is more flexible for metamodeling because it permits the estimation of main effects and two-way interactions. The analyst assumes that these dominate any higher-order interactions. The '5' indicates that main effects are confounded with 4-way interactions ($1+4=5$) and two-way interactions are confounded with three-way interactions ($2+3=5$), but main effects and two-way interactions are unconfounded ($1+2<5$).

To provide some visual intuition for how this works, consider the 2^{3-1} R3FF in Figure 3.8. This is a saturated design for main effects only. The dps in Figure 3.8a are found as labels for the corresponding corners in Figure 3.8b. Note the elegant symmetry: each face of the cube has exactly two dps, enabling us to estimate the slope as the difference between the average of the responses on opposite faces. For example, subtracting the average of responses at points 1 and 3 from the average of the responses at points 2 and 4 gives an estimate of the effect of factor X_1, and so on. Similarly, symmetry and balance are achieved (but more difficult to visualize) for fractional factorial designs in four or more dimensions.

Design Point	Factor (was)	X_1	X_2	X_3 (X_1X_2)
1		−	−	+
2		+	−	−
3		−	+	−
4		+	+	+

(a) Design in coded units.



(b) 3-D plot of numbered dps.

Figure 3.8: 2^{3-1} resolution III fractional factorial (R3FF).

Saturated or nearly-saturated fractional factorials are very efficient when we have lots of factors. For example, 64 design points could be used for a single replication of a 64-dp design involving 62 factors, or two replications of a 32-dp design involving 31 factors. Such designs, like the one in Table 3.8, would be R3FF designs because there will not be enough distinct points to fit interaction terms.

However, our decades of experience in running and analyzing simulation experiments indicates that interactions are almost always present. Interactions are extremely important for decision makers to know about, since they affect our understanding of cause and effect, and therefore may modify the decisions. For these reasons, if you insist on using fractional factorial designs for a simulation study we recommend that, at a minimum, you use an R5FF

instead of an R3FF. The number of dps will be larger, but that may be preferable to foregoing the chance to estimate interactions and R5FF designs are still far more efficient than full factorial designs. The number of metamodel terms in your analysis provides a lower bound for how many degrees of freedom must be available, and the growth rate for these is $O(k^2)$ (quadratic) vs $O(2^k)$ (exponential) for R5FF and full factorial designs, respectively. (See Appendix D for a refresher on the notation of computational complexity.)

Looking back at Table 3.6, if we varied a new factor X_5 at the levels corresponding to the $X_1X_2X_3X_4$ interaction, we would have a perfectly orthogonal design capable of simultaneously estimating all main effects and two-way interactions. This would yield a 2^{5-1} R5FF. Other examples for small k are provided in Table 3.9. For full factorials, the maximum number of terms is equal to the number of design points so a separate column is not provided, but these designs must be replicated or the highest-order interaction ignored in order to have at least one d.f. for estimating error. For more discussion of data farming (i.e., large-scale experiments) involving fractional factorials, see Section 3.3.7.

Table 3.9: Comparison of fractional and full factorial design sizes (numbers of design points) for small experiments ($k \leq 10$). We do not recommend R3FF for data farming experiments due to their severe limitations on identifying interactions. Asterisks denote designs that must be replicated in order to have at least one d.f. for estimating error.

	R3FF			R5FF			Full factorial	
k	# DPs	max # terms	notation	# DPs	max # terms	notation	# DPs	notation
2	4	2	2^{2-0}	4*	3	2^{2-0}	4	2^2
3	4*	3	2^{3-1}	8	6	2^{3-0}	8	2^3
4	8	4	2^{4-1}	16	10	2^{4-0}	16	2^4
5	8	5	2^{5-2}	16	15	2^{5-1}	32	2^5
6	8	6	2^{6-3}	32	21	2^{6-1}	64	2^6
7	8*	7	2^{7-4}	64	28	2^{7-1}	128	2^7
8	16	8	2^{8-4}	64	36	2^{8-2}	256	2^8
9	16	9	2^{9-5}	128	45	2^{9-2}	512	2^9
10	16	10	2^{10-6}	128	55	2^{10-3}	1,024	2^{10}

3.3.5 Central Composite Designs

helpful datafarming gem scripts

`generate_design.rb -d ccd` *`options/range-specs`*
generate resolution V central composite designs (R5CCDs) with scaling

2^k factorials and 2^{k-p} fractional factorials sample each factor at only two levels, so they are

very efficient at identifying slopes for main effects and two-way interactions. The downside of sampling at only 2 levels is that the analyst has no idea about what's happening anywhere in the middle of the factor ranges. Increasing sampling to a 3^k factorial would let us estimate quadratic effects, but (as Table 3.7 previously showed) would require substantially more experiments—especially if k is large. Our goal is to create a design which has 3 levels per factor but having far fewer design points than a brute-force 3^k factorial.

An alternative "classic" design that lets us estimate all full second-order models (i.e., main effects, two-way interactions, and quadratic effects) is called a *central composite design* (usually abbreviated as CCD). Figure 3.9 visually illustrates how a CCD is constructed for $k = 3$ factors. We begin by using either a full 2^k factorial or a 2^{k-p} fractional factorial. We then supplement that basic design with a set of additional design points often referred to as "center and star points." The center point appears at the center of the design space, and the star points are comprised of one point at the center of each face of the cube/hypercube corresponding to the factorial portion of the design. A different way of thinking about it is that each factor has its own axis—at right angles to all the other factor's axes—and you place a design point at the high and low ends of each axis, along with the center point where all the axes intersect.

As a more general example, let's consider a design with standard ± 1 coding, i.e., -1 and $+1$ are the low and high levels, respectively. The center point then occurs at $(0, 0, ..., 0)$; the first axis's star points are $(-1, 0, ..., 0)$ and $(+1, 0, ..., 0)$; the second axis's star points

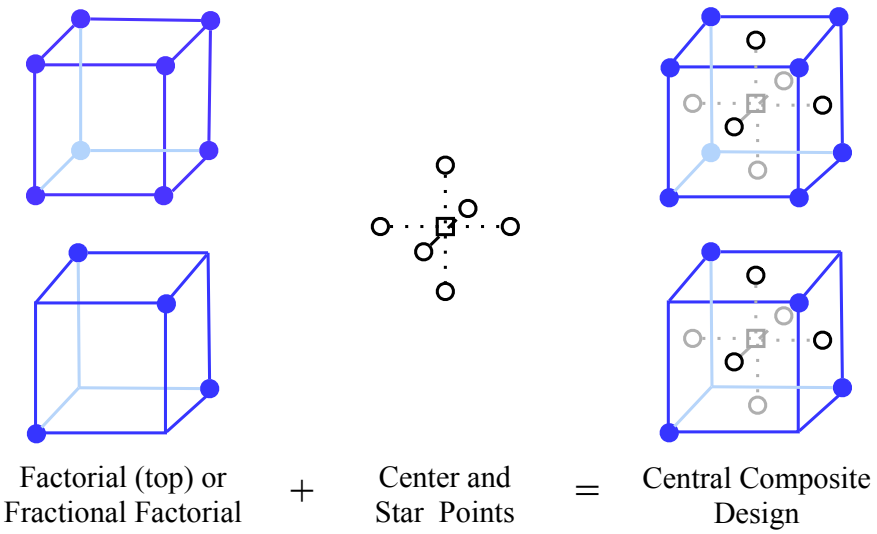


Figure 3.9: Central composite designs (CCDs) for $k = 3$ factors. The top uses a full factorial to construct an R7CCD, permitting simultaneous estimation of all main effects, quadratic effects, two-way interaction effects, and the single three-way interaction. The bottom uses an R3FF to construct an R3CCD, permitting simultaneous estimation of three main effects and three quadratic effects, but the main effects are correlated with the two-way interactions.

are $(0, -1, 0, ..., 0)$ and $(0, +1, 0, ..., 0)$; and so on. After some thought, it's easy to see that for k factors the star and center points will create $2k + 1$ new design points in addition to the 2^k or 2^{k-p} points from the factorial portion of the CCD. If we start with a full factorial, adding the star and center points to create a CCD brings the total number of design points to $2^k + 2k + 1$.

Big-O analysis tells us that as k gets large, the total cost is dominated by the factorial portion of the design, so the additional cost of a CCD is relatively negligible and far less than obtaining three levels per factor by running a 3^k factorial design. In other words, although the CCD adds more points when there are more factors, the basic design is still much smaller than a 3^k for any given k. Table 3.10 shows that the savings in design points can be dramatic if fractional factorial designs are used instead of full factorials. Note that we report design sizes for both resolution 3 (main effects only) and resolution 5 (main effects and two-way interactions), although we highly recommend the use of resolution 5 designs. We remark that the CCDs and 3^k factorials can also detect nonlinear effects (e.g., quadratic terms in polynomial regression metamodels).

Table 3.10: Comparison of classical design sizes for small experiments ($k \leq 10$). We do not recommend R3FF or R3CCD for data farming experiments due to their limitations on identifying interactions. R3FF designs marked with asterisks must be replicated to fit full main-effects metamodels.

	# Design Pts						
k	R3FF	R3CCD	R5FF	R5CCD	2^k	3^k	5^k
2	4	7	4	9	4	9	25
3	4*	11	8	15	8	27	125
4	8	13	16	25	16	81	625
5	8	19	16	27	32	243	3,125
6	8	21	32	45	64	729	15,625
7	8*	23	64	79	128	2,187	78,125
8	16	25	64	81	256	6,561	390,625
9	16	35	128	147	512	19,683	1,953,125
10	16	37	128	149	1,024	59,049	9,765,625

Although CCDs can be based on resolution III, resolution V, or full factorial designs, through the remainder of this book we will use the term 'CCD' to refer to a resolution 5 CCD unless stated otherwise.

Each subplot of a scatterplot matrix for a CCD will appear identical, just as each subplot of the scatterplot matrices for full or fractional factorials is identical. As with the full or fractional factorial designs which CCDs use as starting points, many points of the multi-dimensional design overlay each other when projected onto two dimensions.

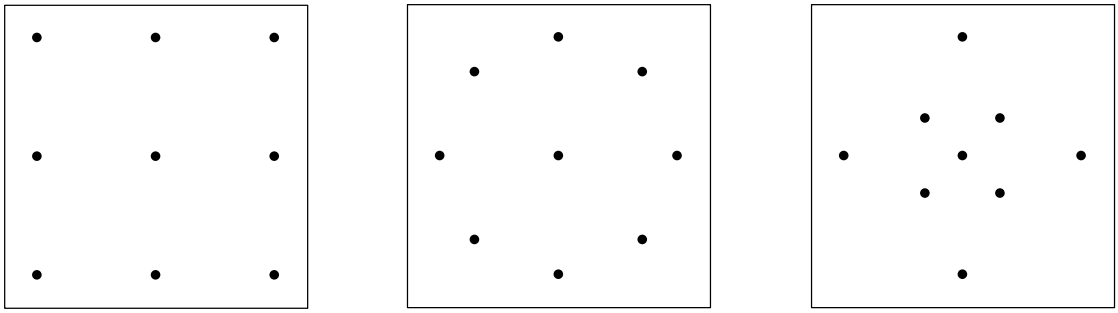
3.3.6 Rotatable Central Composite Designs

helpful datafarming gem scripts

`generate_design.rb -d rotatable` *`options/range-specs`*
generate rotatable resolution V CCDs with scaling

The numbers of design points in Table 3.10 align with those for designs generated using the `generate_design.rb` ruby script. The table entries are obtained for each value of k by summing the numbers of design points in a corresponding R5FF, the number of star points ($2k$), and a single center point.

However, there is one more twist involving CCDs that has to do with the distance of the star points from the center point. For the face-centered CCDs we've described so far, each subplot will be a grid of 9 points as in Figure 3.10a, regardless of k. While this is nice and



(a) Face-centered CCD for all k **(b)** Rotatable CCD for $k = 2$. **(c)** Rotatable CCD for $k = 10$.

Figure 3.10: Scatterplot patterns for various CCDs

regular, the corner points geometrically are farther from the center than the star points. If we make all the radii identical the result is called a rotatable CCD, and is another option of the `generate_design.rb` script. We have opted to make CCDs rotatable by pulling the corner points in rather than extending the star points out to preserve the maximum and minimum limits of the ranges chosen by the analyst. When $k = 2$, as in Figure 3.10b, it is visually clear that the center point is equidistant from the other 8 combinations of any two factors. For larger k the corner points are the same distance from the center as the star points, but appear closer to the center in two dimensional projections as illustrated by Figure 3.10c with $k = 10$. The face-centered and rotatable CCDs have the same design sizes, but the rotatable CCDs can distinguish between other types of nonlinearity because each factor is explored at 5 distinct levels.

One potential disadvantage of rotatable CCDs for large k is that they have far fewer dps that have one or more factors set near their minimum or maximum value. One advantage of rotatable CCDs is the additional flexibility in metamodel construction, since each factor is sampled at five levels instead of three.

3.3.7 Classical Design Options for Data Farming Experiments

helpful datafarming gem scripts

`generate_design.rb -d ccd` *`options/range-specs`*
generate CCDs with scaling and stacking

`generate_design.rb -d rotatable` *`options/range-specs`*
generate rotatable CCDs with scaling and stacking

When we move to a data farming environment, we are "thinking big" in terms of the number of factors and the flexibility when analyzing the results.

Use factorials judiciously

As indicated in Section 3.3.3 and Table 3.10, m^k full factorial designs are unwieldy even for moderate k (say, $k = 10$) when there are $m \geq 3$ levels per factor, and are untenable if we are interested in dozens or hundreds of factors. Consequently, full factorial designs are unsuitable for large-scale data farming experiments. However, they may have some value in special circumstances. Specifically,

- A full factorial for a small number of factors may be crossed with a different type of design (e.g., R5FF, CCD, or rotatable CCD) for the remaining factors. This may be particularly useful if we have a few categorical factors involving a small number of levels. For example, you could extend the gummy bear experiment to consider 4 operators and 2 colors of gummy bears, in addition to the number of books (discrete) and the launch position. The crossed design would be a $2^2 3^1 4^1$ full factorial with 48 dps. If you chose to experiment with 5 launch positions instead of 2, the design would be a $2^1 3^1 4^1 5^1$ full factorial with 120 dps.

  More generally, you're likely to have factors of many types (continuous-valued, discrete-valued, and qualitative) in a large-scale data farming study. Factorials can be constructed for any type of factor. R5FF or 2^k are suitable for binary-valued factors, whether discrete or continuous. CCDs are suitable for continuous-valued factors, but may require too much rounding to work well for discrete-valued factors with a small number of levels, especially in the case of rotatable CCDs.
- A full factorial might be useful during later iterations of the data farming process. If you have already conducted an experiment involving large k, and whittled that down to the most important terms that involve a handful of factors or less ($k' \ll k$) via metamodels (see Chapter 4), then you could consider crossing two designs: (i) a $3^{k'}$ or $4^{k'}$ for the k' quantitative factors deemed most important, and (ii) a design with a single dp holding all other $k - k'$ factors at their nominal or center point values.

- Similar to the previous bullet, you could consider crossing two designs: (i) a $3^{k'}$ or $4^{k'}$ for the k' factors deemed most important, and (ii) a design with the remaining $k - k'$ factors held at nominal or center point values.

Large-scale fractional factorials, CCDs, and rotatable CCDs

R5FFs, CCDs, and rotatable CCDs are possible designs even for large k if we base the CCDs on fractional factorials instead of full factorials. These resolution V CCDs were created using a computational approach, and increased the number of factors for which designs are readily available by an order of magnitude, from $k \leq 12$ factors to $k \leq 120$ factors. Selected designs are presented in Table 3.11. The values of k are associated with the lowest and highest number of factors possible for a given size of the fractional factorial component. For intermediate entries, simply add an additional 2 design points for each additional factor. For example, since the design for $k = 18$ has 549 dps, we know that the design for $k = 19$ will have $549 + 2 = 551$ dps and that for $k = 20$ will have 553 dps. The k values for which the fractional factorial portion of the design jumps to the next power of two are not easily predictable, which is why we have provided the table. The CCD design sizes in Table 3.11 are one smaller than those reported in the original paper (Sanchez and Sanchez, 2005) because the newer designs do not duplicate the center points.

Table 3.11: R5FF and R5CCD design Sizes for k factors

	# Design Pts			# Design Pts			# Design Pts	
k	R5FF	R5CCD	k	R5FF	R5CCD	k	R5FF	R5CCD
2	4	9	12	256	281	39	4,096	4,175
3	8	15	17	256	291	52	4,096	4,201
4	16	25	18	512	549	53	8,192	8,299
5	16	27	21	512	555	69	8,192	8,331
6	32	45	22	1,024	1,069	70	16,384	16,525
7	64	79	29	1,024	1,083	92	16,384	16,569
8	64	81	30	2,048	2,109	93	32,768	32,955
9	128	147	38	2,048	2,125	120	32,768	33,009
11	128	151						

One benefit of the design generation approach used by Sanchez and Sanchez (2005) is that there is a very compact notation for describing the design. The designs are specified using a discrete-valued orthogonal function set called Walsh functions—a binary-valued equivalent to sinusoidal functions—each referenced by a unique index called its sequency. Conceptually, rather than storing 119 different design matrices for the designs of Table 3.11, we need only store one list of 120 sequencies $S \equiv \{s_1, s_2, \ldots, s_{120}\}$ indexed by k, corresponding to the sequencies at which to vary each of the 120 factors. From the list S, it is possible to derive

another implicit list indicating the fractional reduction p used for the R5FF size 2^{k-p} by computing $p = k - \lceil \log_2(S_k + 1) \rceil$.

$$k = \{1,2,3,4,\ 5,\ 6,\ 7,\ 8,\ 9,10,\ 11,\ 12,\ 13,\ 14,\ 15,\ 16,\ 17,\ldots,\ 120\}$$
$$S = \{1,2,4,8,15,16,32,51,64,85,106,128,150,171,219,237,247,\ldots,32705\}$$
$$p = \{0,0,0,0,\ 1,\ 1,\ 1,\ 2,\ 2,\ 3,\ \ 4,\ \ 4,\ \ 5,\ \ 6,\ \ 7,\ \ 8,\ \ 9,\ldots,\ \ 105\}$$

For example, a 2^{6-1} design is generated by varying the first 6 factors at sequencies 1, 2, 4, 8, 15, and 16 using a design size of $2^{6-1} = 32$; a 2^{8-2} design is generated by varying the first 8 factors at sequencies 1, 2, 4, 8, 15, 16, 32, and 51 using a design size of $2^{8-2} = 64$; a $2^{120-105}$ design is generated by varying all 120 factors according to the list using a design size of $2^{120-105} = 32768$. Note that the datafarming gem will generate the resolution V fractional factorials, CCDs, or rotatable CCDs for any $k \in \{2, 3, \ldots, 120\}$. Designs for any $k \in \{121, 122, \ldots, 443\}$ are available from the authors of this book on request.

We remark that some of the R5FF and R5CCDs in Table 3.11 are actually resolution 5+, indicating that all main effects, all two-way interactions, and *some* of the three-way interactions can simultaneously be estimated. However, as we mentioned earlier, assume that the notation CCD or rotatable CCD refers to a resolution 5 or 5+ CCD in the context of data farming experiments.

Stacking for additional design points

Replication yields the same set of dps multiple times. If the goal is to generate more data while increasing the number of dps, then *stacking* the design can either supplement or substitute for pure replication. Stacking works by concatenating two or more designs to obtain a larger design. For example, by stacking two different R5FFs we may pick up additional corner points in the k-dimensional space.This provides some additional flexibility when it comes time to analyze the output from our experiment. Stacking two designs means running both sets of design points. One way to obtain two different designs from the same R5FF matrix is to reassign the factors to different columns of the experiment design matrix. We show this for an R5FF with $k = 6$ in Table 3.12. The initial stack appears in Table 3.12a. In Table 3.12b, we shift the initial factor names over to the right, and then move the last factor name back to the first column. In order to line up the factor levels for stacking, we end up shifting the new first column (for factor X_6) all the way over to the far right in Table 3.12c.

We call this reassignment scheme *shift and stack*, and `generate_design.rb` allows you to do this for any of the designs implemented. For the R5FFs, once $k \geq 6$ you will generate new dps that were not in the initial stack, even though many of the dps repeat. To see this, note that the first dp in Table 3.12a has all factors set to their high levels, so reassigning factors to columns simply duplicates this dp in Table 3.12c. In contrast, the second dp in Table 3.12c has X_4 and X_6 set to their low levels and the other four factors set to their high levels. A close inspection of Table 3.12a shows this dp is not part of the initial stack. If $k = 6$, the

Table 3.12: Creating a second stack from an R5FF with $k = 6$ and 32 design points. The initial stack is shown in 3.12a. In 3.12b, we shift the initial factor names over to the right, and then move the last factor name back to the first column. In order to line up the factor levels for stacking, we end up shifting the new first column (for factor X_6) all the way over to the far right in 3.12c, where new dp configurations are marked with an asterisk.

(a) Initial stack

X_1	X_2	X_3	X_4	X_5	X_6
1	1	1	1	1	1
-1	1	1	1	-1	1
1	-1	1	1	-1	1
-1	-1	1	1	1	1
1	1	-1	1	-1	1
-1	1	-1	1	1	1
1	-1	-1	1	1	1
-1	-1	-1	1	-1	1
1	1	1	-1	-1	1
-1	1	1	-1	1	1
1	-1	1	-1	1	1
-1	-1	1	-1	-1	1
1	1	-1	-1	1	1
-1	1	-1	-1	-1	1
1	-1	-1	-1	-1	1
-1	-1	-1	-1	1	1
1	1	1	1	1	-1
-1	1	1	1	-1	-1
1	-1	1	1	-1	-1
-1	-1	1	1	1	-1
1	1	-1	1	-1	-1
-1	1	-1	1	1	-1
1	-1	-1	1	1	-1
-1	-1	-1	1	-1	-1
1	1	1	-1	-1	-1
-1	1	1	-1	1	-1
1	-1	1	-1	1	-1
-1	-1	1	-1	-1	-1
1	1	-1	-1	1	-1
-1	1	-1	-1	-1	-1
1	-1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1

(b) Reassigning factors to columns

X_6	X_1	X_2	X_3	X_4	X_5
1	1	1	1	1	1
-1	1	1	1	-1	1
1	-1	1	1	-1	1
-1	-1	1	1	1	1
1	1	-1	1	-1	1
-1	1	-1	1	1	1
1	-1	-1	1	1	1
-1	-1	-1	1	-1	1
1	1	1	-1	-1	1
-1	1	1	-1	1	1
1	-1	1	-1	1	1
-1	-1	1	-1	-1	1
1	1	-1	-1	1	1
-1	1	-1	-1	-1	1
1	-1	-1	-1	-1	1
-1	-1	-1	-1	1	1
1	1	1	1	1	-1
-1	1	1	1	-1	-1
1	-1	1	1	-1	-1
-1	-1	1	1	1	-1
1	1	-1	1	-1	-1
-1	1	-1	1	1	-1
1	-1	-1	1	1	-1
-1	-1	-1	1	-1	-1
1	1	1	-1	-1	-1
-1	1	1	-1	1	-1
1	-1	1	-1	1	-1
-1	-1	1	-1	-1	-1
1	1	-1	-1	1	-1
-1	1	-1	-1	-1	-1
1	-1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1

(c) Second stack, columns reordered

X_1	X_2	X_3	X_4	X_5	X_6
1	1	1	1	1	1
1	1	1	-1	1	-1*
-1	1	1	-1	1	1
-1	1	1	1	1	-1*
1	-1	1	-1	1	1
1	-1	1	1	1	-1*
-1	-1	1	1	1	1
-1	-1	1	-1	1	-1*
1	1	-1	-1	1	1
1	1	-1	1	1	-1*
-1	1	-1	1	1	1
-1	1	-1	-1	1	-1*
1	-1	-1	1	1	1
1	-1	-1	-1	1	-1*
-1	-1	-1	-1	1	1
-1	-1	-1	1	1	-1*
1	1	1	1	-1	1*
1	1	1	-1	-1	-1
-1	1	1	-1	-1	1*
-1	1	1	1	-1	-1
1	-1	1	-1	-1	1*
1	-1	1	1	-1	-1
-1	-1	1	1	-1	1*
-1	-1	1	-1	-1	-1
1	1	-1	-1	-1	1*
1	1	-1	1	-1	-1
-1	1	-1	1	-1	1*
-1	1	-1	-1	-1	-1
1	-1	-1	-1	-1	1*
1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1*
-1	-1	-1	-1	-1	-1

initial stack has 32 distinct dps, and the number of distinct design points for s stacks is 48, 56, 60, 62, and 63 ($s = 2, 3, \ldots, 6$). The growth rate of new dps is closer to linear for larger k when p is large. For instance, when $k = 30$, the initial stack has 2048 dps and the number of distinct design points for s stacks is 4095, 6142, 8189, and 10235 ($s = 2, 3, 4, 5$), or 61381 for 30 stacks.

A comprehensive set of stacks would utilize every permutation of the columns in the base design. The `generate_design.rb` program does not attempt to be comprehensive, but uses the shift-and-stack scheme as a convenient way of generating additional design points.

Stacking could be done on subdesigns that are crossed to get a larger design. For instance, in a study of a queuing system, our first design could explore a single qualitative factor for the queueing discipline: first-in-first-out (FIFO), last-in-first-out (LIFO), or shortest-

processing-time first (SPT). Our second design could be a rotatable CCD with more than one stack for several quantitative factors, such as the number of servers, service time mean, service time standard deviation, arrival rate, server breakdown rate, and server downtime distributional parameters. Crossing these two designs would produce a larger—but still orthogonal—design. Since we know that queueing systems have non-homogeneous variance, we should use replication to be able to estimate variability in the responses.

Finally, it is possible to stack different types of designs, such as a R5FF for 50 factors along with an 11^3 full factorial involving three particular factors of interest while holding all others at their center value—this would still be an orthogonal design as long as the center levels remain the same for the three special factors.

3.4 Modern Space-filling Designs

This section introduces you to 21^{st} century designs, which have been developed specifically with data farming in mind. The classical designs were parsimonious in terms of the number of design points, but bear in mind that a parsimonious design is only useful if it can capture the underlying response surface. A design that can't do so is a false economy, the term economists coined for an attempt at savings that ends up costing more.

Once we move into the realm of computerized models, it is both obvious and natural to consider automating the process of running experiments. Automation unlocks the ability to expand the size of the designs, and this is further multiplied by the growth of parallel processing. Simultaneously, simulations or other computerized models of even moderate complexity have more inputs/factors than earlier generations of designs can accommodate. Together, these observations have been compelling motivations for the development of efficient large scale designs of experiments.

The designs presented here are built on the foundations of prior work.

- Hand-constructed and randomized Latin Hypercubes date back to the 1980's, but Nearly Orthogonal Latin Hypercubes (Section 3.4.2), a.k.a. NOLHs, extended the concept by offering guaranteed bounds on the maximum pairwise correlation *by construction* of the design's columns.
- NOLHs were designed for continuous-valued factors. Although rounding seems like a natural approach for dealing with discrete-valued factors, doing so often leads to higher correlations than desired or lack of balance in the frequency with which factor levels occur. Designs called orthogonal arrays have been developed to ensure balance for categorical factors, but applying them in a study involving a mixture of discrete, categorical, and continuous factors has traditionally involved crossing different classes of designs, resulting in rapid growth of the overall design's size. Nearly orthogonal-and-balanced designs (NOB/NOABs) were created to generate mixed factor designs which simultaneously maintain low correlation amongst all quantitative factors, avoid

large imbalances for the discrete or categorical factors, and accomplish both these goals efficiently in a reasonable number of dps.

- The use of Chebyshev points, which are based on sampling at discrete Fourier frequencies, offers improved polynomial fitting for single factor experiments. Work done in the 1980's also leveraged the orthogonality of discrete Fourier frequencies to identify significant factors in the presence of serially correlated time series using spectral analysis. Frequency-Based Designs (Section 3.4.4), a.k.a. FBDs, extend the concept to generate efficient and perfectly orthogonal designs capable of fitting full second order metamodels, i.e., main effects, quadratics, and two-way interactions.
- Frequency-Based Screening designs (Section 3.4.5) leverage the mathematics of FBDs, substantially reduce the design size by allowing partial correlation of interactions while retaining full identifiability.

These designs offer many important benefits. They enable studies of many factors at many levels, which is necessary for simulations where large numbers of potential factors exist. Their space-filling nature also provides analysis flexibility, which facilitates the construction of more complex metamodels (e.g., fitting both polynomial and non-polynomial nonlinearity) and improves your ability to detect and diagnose lack-of-fit in those metamodels.

3.4.1 Background: Random Latin Hypercubes

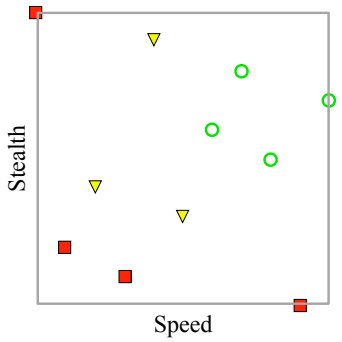
Latin hypercube (LH) designs provide a flexible way of constructing efficient designs for quantitative factors. They have some of the space-filling properties of factorial designs with fine grids, but require orders of magnitude less sampling. Once again, let k denote the number of factors, and let $m \geq k$ denote the number of design points. The factor levels can be coded as m equally-spaced values in the range $\{-1, 1\}$ as $\{\frac{2i-(m+1)}{m-1} : i = 1 \ldots m\}$. However, a more common way of denoting factor levels for LH designs is to just use the integers $\{1 \ldots m\}$. In either case, a *random LH design* has a random permutation of the m values for each column of the design matrix, independent of the permutations of other columns. Random LH designs can be constructed for any number of factors k provided that $m \geq k$, but collinearity problems often arise unless $m \gg k$.

Figure 3.11 lists a random LH with $k = 2$ and $m = 11$, and provides a picture of results that might arise by using this experiment design for our capture-the-flag simulation. (Note that the design matrix has been split into halves, which are placed side-by-side to reduce the vertical height of the figure.) Compare this design to those of Figure 3.5. Unlike the 2^2 factorial design, the LH design provides some information about what happens in the center of the experimental region, but requires far fewer experiments than the 11^2 factorial design.

As already mentioned, random LH designs generally have good orthogonality properties if m is much larger than k, but for smaller designs it's common for some factors to have high pairwise correlations. This is illustrated by Figure 3.12, which shows a few of the other 17! (17 factorial, which is over 39.9 million) potential LH's with $k = 2$ and 11 design points.

Speed	Stealth
1	11
3	5
7	7
2	3
5	10
6	4
10	1
4	2
11	8
8	9
9	6

(a) Design matrix.



(b) Scatterplot.

Figure 3.11: Random Latin hypercube for capture-the-flag simulation.

We can visually see that some convey much less information about the response behavior. There are large swaths of the design space lacking design points, and several of the designs show strong linear relationships between the two factors.

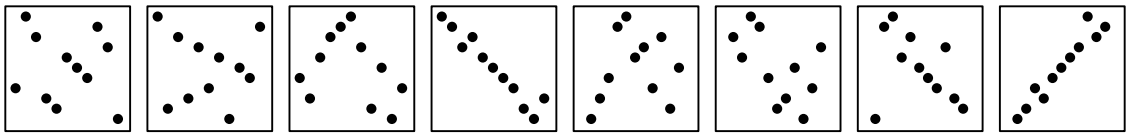


Figure 3.12: Additional random Latin hypercubes for capture-the-flag simulation.

3.4.2 Nearly Orthogonal Latin Hypercubes

helpful datafarming gem scripts

`generate_design.rb -d nolh` *`options/range-specs`*
generate designs by reassigning columns from pre-tabled NOLHs, with factor scaling and stacking, for up to 100 factors

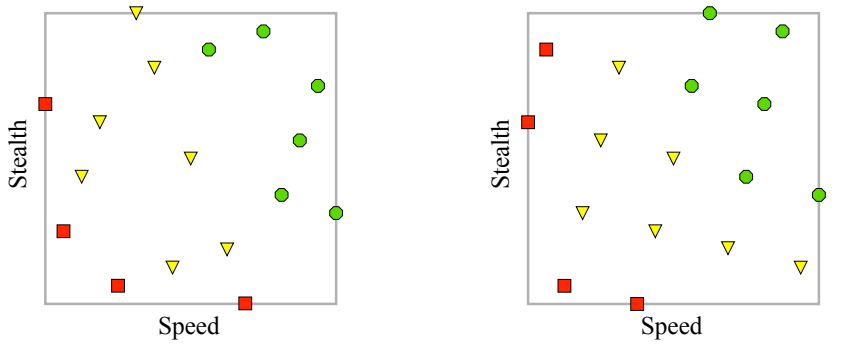
One approach to obtaining a good LH design is to generate many random LH designs and then choose a good one. (This is actually what we did for Figure 3.11). Alternatively, we can use a tabled design—that is, we rely on the hard work that someone else has already done to create a good design. Cioppa and Lucas (2007) constructed *nearly orthogonal Latin hypercube* (NOLH) designs that have good space-filling and orthogonality properties for small or moderate k (up to $k = 22$). Table 3.13 lists the number of design points for NOLHs with $k \leq 100$. These designs are not square, but the number of design points for a given k is

Table 3.13: Data requirements for Nearly Orthogonal Latin Hypercube designs created using `generate_design.rb`

Max Number of Factors	Number of Design Points
7	17
11	33
16	65
22	129
29	257
100	512

radically less than the numbers for the gridded designs in Table 3.7. For example, 20 factors can be explored in an NOLH with only 129 design points, as compared to over one million design points needed for a 2^{20} factorial. The design for $\max(k) = 29$ is from Hernandez et al. (2012), and the design for $max(k) = 100$ is the continuous factors portion of a combined continuous/discrete design from Vieira Jr et al. (2013).

Figure 3.13 provides pictures of capture-the-flag results that might arise by using the smallest NOLH, with 17 design points. Figure 3.13a uses the design's first column for speed and second column for stealth, while Figure 3.13b uses the first and third columns for speed and stealth, respectively.



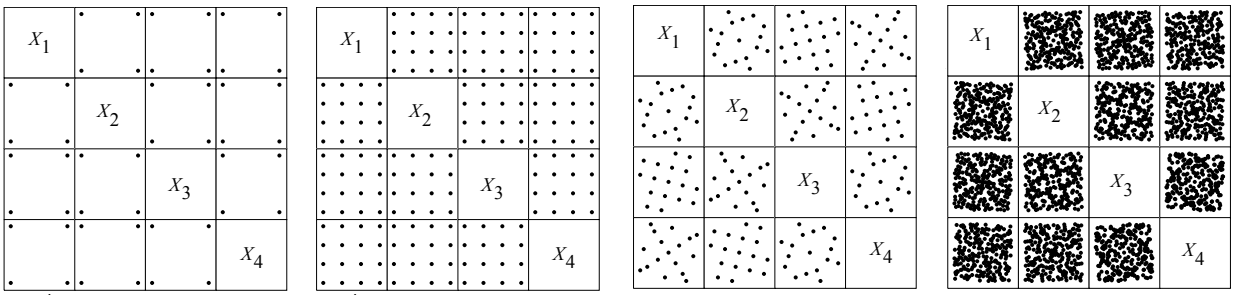
(a) NOLH (columns 1 and 2) for capture the flag

(b) NOLH (columns 1 and 3) for capture the flag

Figure 3.13: Nearly orthogonal Latin hypercubes with 17 design points for capture the flag.

Scatterplot matrices of four different designs are shown in Figure 3.14. These are a 2^4 factorial design, a 4^4 factorial design, an NOLH design with 17 design points, and an NOLH design with 257 design points. The two-dimensional space-filling behavior of the NOLH in Figure 3.14(c) compares favorably with that of the 4^4 factorial for roughly 1/15 the computational effort, so experimenters concerned about the level of computational effort might prefer the latter. Alternatively, experimenters considering the use of the 4^4 factorial, and thus willing to run experiments at 256 design points, might prefer the NOLH in Figure 3.14(d) with

257 design points (just one more)—and gain the ability to examine a much denser set of factor-level combinations, as well as explore up to 25 additional factors without the need for additional sampling!



a. 2^4 factorial: 16 design points b. 4^4 factorial: 256 design points c. NOLH: 17 design points d. NOLH: 257 design points

Figure 3.14: Scatterplot Matrices for Selected Factorial and NOLH Designs

The efficiency of LH sampling is greatest for large k. If running our simulation model at a single design point takes one second, then each replication of a 29-factor NOLH design would take under five minutes, versus over 17 years per replication if we use a 2^{29} full factorial design. This massive gain in efficiency allows us to perform multiple replications of the design. Replication is highly recommended. It improves the precision of our metamodel parameter estimates but it also enables us to assess whether or not constant error variance is a reasonable characterization of the simulation's performance.

Stacked Latin Hypercubes

helpful datafarming gem scripts

`generate_design.rb -d nolh` *`options/range-specs`*
generate designs by reassigning columns from pre-tabled NOLHs, with factor scaling and stacking, for up to 100 factors

If we have the time and budget for even more sampling, then two or more different Latin hypercubes or NOLHs can be *stacked* to obtain a larger design with better space-filling properties. Stacking two designs means running both sets of design points. One way to obtain two different designs from the same NOLH matrix is to reassign the factors to different columns of the experiment design matrix, as we illustrated in Figure 3.13. Let's explore this by looking at a 17-dp OLH. (Yes, we meant OLH rather than NOLH because until we start rounding levels, this particular LH is perfectly orthogonal.)

The basic design, in coded levels, appears in Table 3.14. At the top, we've listed the factor names associated with each column. If we just used a single design, this is it. If we want to shift and stack several designs, this is the first one in the stack.

Table 3.14: Orthogonal LH with 17 design points

X_1	X_2	X_3	X_4	X_5	X_6	X_7
6	17	14	7	5	16	10
2	5	15	10	1	6	11
3	8	2	5	11	14	17
4	11	6	17	10	3	13
13	16	8	3	6	1	14
17	6	7	14	2	13	15
11	4	17	6	15	8	16
10	15	13	16	14	11	12
9	9	9	9	9	9	9
12	1	4	11	13	2	8
16	13	3	8	17	12	7
15	10	16	13	7	4	1
14	7	12	1	8	15	5
5	2	10	15	12	17	4
1	12	11	4	16	5	3
7	14	1	12	3	10	2
8	3	5	2	4	7	6

Table 3.15: Creating a second stack from the Orthogonal LH with 17 design points. On the left, we shift the initial factor names over to the right, and then move the last factor name back to the first column. In order to line up the factor levels for stacking, we end up shifting the new first column (for factor X_7) all the way over to the far right.

(a) Reassigning factors to columns

X_7	X_1	X_2	X_3	X_4	X_5	X_6
6	17	14	7	5	16	10
2	5	15	10	1	6	11
3	8	2	5	11	14	17
4	11	6	17	10	3	13
13	16	8	3	6	1	14
17	6	7	14	2	13	15
11	4	17	6	15	8	16
10	15	13	16	14	11	12
9	9	9	9	9	9	9
12	1	4	11	13	2	8
16	13	3	8	17	12	7
15	10	16	13	7	4	1
14	7	12	1	8	15	5
5	2	10	15	12	17	4
1	12	11	4	16	5	3
7	14	1	12	3	10	2
8	3	5	2	4	7	6

$\rightarrow$

(b) Second stack, columns re-ordered.

X_1	X_2	X_3	X_4	X_5	X_6	X_7
17	14	7	5	16	10	6
5	15	10	1	6	11	2
8	2	5	11	14	17	3
11	6	17	10	3	13	4
16	8	3	6	1	14	13
6	7	14	2	13	15	17
4	17	6	15	8	16	11
15	13	16	14	11	12	10
9	9	9	9	9	9	9
1	4	11	13	2	8	12
13	3	8	17	12	7	16
10	16	13	7	4	1	15
7	12	1	8	15	5	14
2	10	15	12	17	4	5
12	11	4	16	5	3	1
14	1	12	3	10	2	7
3	5	2	4	7	6	8

Now, let's go ahead and see how to add more stacks. On the left of Table 3.15 we show a different assignment of factor names to columns. We've gone ahead and highlighted one row in red: this is a center point, and will not change if we reassign factors to different columns. Including or excluding this center point after the initial stack has no effect on the design's orthogonality, and is one of the options available in `generate_design.rb`.

Looking at Table 3.15, it's clear that we have different combinations of factor settings—in other words, we now have different sets of design points other than the center point, as we noted earlier. For example, in the initial stack (Table 3.14), two of the combinations of X_1 and X_2 are $\{1, 12\}$ and $\{2, 5\}$. In the second stack (Table 3.12(b)), we have $\{1, 4\}$ and $\{2, 10\}$. We now have *two different* values of X_2 associated with each value of X_1, except the center point.

To see how this looks graphically, the four sub-plots of Figure 3.15 show the results of shifting and stacking. In (a), we have the initial stack. In (b), we show the second stack. Note that the patterns initially showing in the $X_1 \times X_2$ scatterplot now show up in the $X_2 \times X_3$ scatterplot, etc. The first two stacks are superimposed in (c), and all seven stacks are shown in (d). Clearly, there is better space-filling as we add stacks.

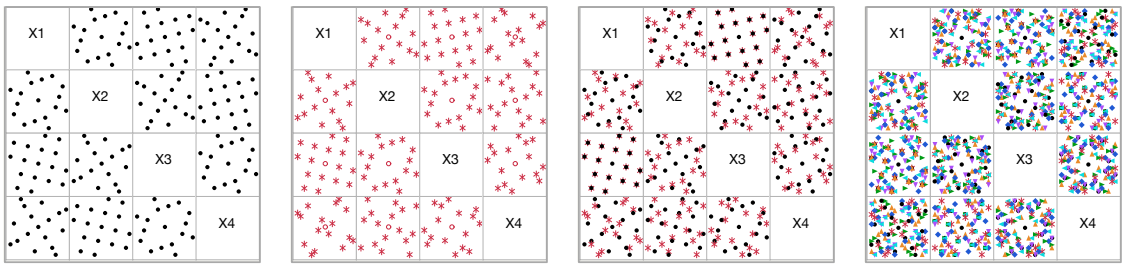
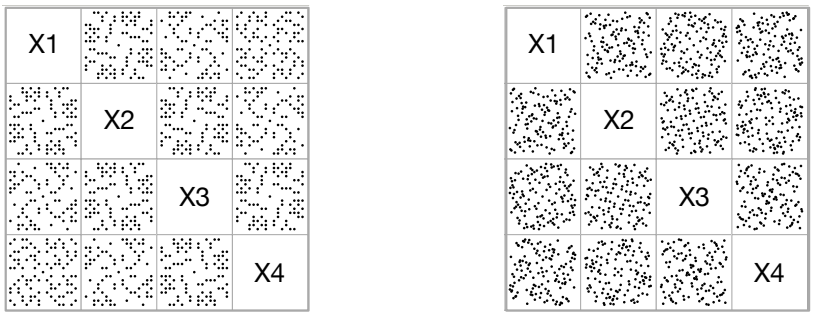


Figure 3.15: Scatterplot Matrices for Shifted and Stacked OLH Design, 17 initial design points.

If we'd known ahead of time that we were going to take more sampling, we could have started with a larger design. Figure 3.16 shows the space-filling properties of two alternatives that are fairly similar in terms of their computational requirement. In Figure 3.16(a) we repeat the results of the 7-stack design of Figure 3.15(d), but with the colors removed to avoid distraction. In Figure 3.16(b) we show the pairwise projections of the NOLH with 129 design points. This does a better job of space filling in two dimensions, because each of the factors takes on 129 distinct levels, instead of only 17 levels. We've lost a little orthogonality from 3.15(d)—the maximum absolute pairwise correlation among the first four factors is $|\rho_{map}| = 0.0052$, and if we had investigated all 22 possible factors, we would have $|\rho_{map}| = 0.0074$. However, these are such small numbers that we do, indeed, have a "nearly orthogonal" design.



(a) 7 stacks of a 17dp OLH **(b)** 1 stack of a 129dp NOLH

Figure 3.16: Scatterplot matrices for two NOLH-based designs.

Second-order Latin Hypercubes

The NOLH-based designs (and other space-filling designs with good orthogonal properties) provide a great deal of analysis flexibility. From a metamodeling perspective, they allow us to fit a full first-order metamodel, a metamodel that includes all main effects and up to $n_d - k$ other terms, or a polynomial metamodel of degree $n_d - 1$ for a single factor, to name a few. Stepwise regression combined with analyst judgment are typically needed to come up with a suitable metamodel that balances parsimony and explanatory power. However, there may be instances where we are interested in the possibility of estimating *full* second-order models but also want more flexibility in the analysis methods and metamodeling techniques we will use than the CCDs, which are not space filling, can provide. One way of accomplishing this for a small number of factors is by using a second-order NOLH (MacCalman et al., 2017). The second-order NOLHs are catalogued for up to 12 factors, and available from the software downloads page at `https://harvest.nps.edu`. Design sizes for those with a maximum absolute pairwise correlation between any two columns in the analysis matrix ($|\rho_{map}|$) of at most 0.05 are shown in Table 3.16. A few other designs that exceed the 0.05 threshold are also provided in the download, including a design for $k = 15$ factors with 1000

Table 3.16: Data requirements for 2nd-order NOLH designs

Number of Factors	Number of Design Points
3	15
4	29
5	39
6	70
7	165
8	300
9	370
10	470
11	630
12	775

dps and $|\rho_{map}| = 0.0586$. These are constructed using an evolutionary algorithm as a search heuristic to find designs with good properties.

These designs are not included in `generate_design.rb` because they have been superseded by more recent developments. A different class of second-order designs that are available for moderate and very large k (up to 1000 factors), are fully orthogonal, and are substantially more efficient than the 2nd-order NOLHs for $k > 6$, are described in Section 3.4.4.

3.4.3 Nearly Orthogonal-and-Balanced Designs

helpful spreadsheet

`NOAB_Mixed_Designs_v4.xlsx`
construct custom designs with a mix of discrete and continuous factors

When discrete-valued factors with limited numbers of levels are present, discretizing by rounding NOLH designs may no longer retain their near-orthogonal properties. The nearly orthogonal-and-balanced (abbreviated NOB or NOAB) mixed designs of Vieira Jr et al. (2013) are suitable in these situations. One general-purpose design with $n_d = 512$ allows for the simultaneous investigation of up to 300 factors, comprised of 20 columns each for factors with discrete number of levels $m \in \{2, 3, \ldots, 11\}$, and an additional 100 columns for continuous-valued factors. The columns for the continuous-valued factors constitute an NOLH. Near-orthogonality, as measured by small $|\rho_{map}|$ between pairs of columns, is preserved for all 300 columns. Both this 512-dp NOAB and a smaller, 128-dp NOAB are available in the spreadsheet `NOAB_Mixed_Designs_v4.xlsx` (Vieira Jr et al., 2022) This design allows for the simultaneous investigation of up to 75 factors, and is comprised of 5 columns each for factors with discrete numbers of levels $m \in \{2, 3, \ldots, 11\}$, and an additional 25 columns for continuous-valued factors.

The discrete-valued columns are also "nearly balanced" in the sense that for each factor the number of design points allocated to any of the factor's levels is not allowed to deviate too far from the counts allocated to the other levels. For example, a perfectly balanced 2-level factor would be sampled at its low level in exactly $n_{dp}/2$ design points, and similarly for its high level. With $n_{dp} = 512$, it is impossible to achieve perfect balance for any m-level factor such that m and 512 are relatively prime—such as m equal to 3, 5, 6, etc.—but the NOB design tries to limit the imbalance as follows. Let ℓ_i denote the number of discrete levels for factor X_i, and assume (for convenience) that these levels are coded as $1, 2, \ldots, \ell_i$. Let $w_{i,j}$ be the number of occurrences of level j in the design. Then the maximum imbalance for factor X_i is defined as

$$\delta_i = \max_{j=1,\ldots,\ell_i} \left| \frac{w_{i,j} - (n_{dp}/\ell_i)}{(n_{dp}/\ell_i)} \right|, \tag{3.5}$$

and the maximum imbalance for the design is then defined as

$$\delta_{max} = \max_{i=1,\ldots,k} \delta_i. \tag{3.6}$$

Values of δ_{max} closer to zero mean the design is closer to being balanced.

Some degree of imbalance is inevitable for certain values of m. However, it is extremely difficult to achieve low imbalance without either sacrificing orthogonality or dramatically increasing the number of design points. The 512-dp NOAB design does a good job trading these off, and has a maximum imbalance of $\delta_{max} = 0.113$ and a maximum absolute pairwise correlation of $\rho_{map} = 0.035$. The 128-dp NOAB has a maximum imbalance of $\delta_{max} = 0.195$ and a maximum absolute pairwise correlation of $\rho_{map} = 0.051$.

While it is not possible to show the full scatterplot matrix for the 512-dp NOAB design on a single printed page, we can show this type of plot for a subset of the factors. Figure 3.17 shows a design for a single i-level discrete factor $i = 1, \ldots, 11$ and two continuous-valued factors. The subplots look like full factorial plots for the discrete-valued factors. Subplots involving both a discrete and continuous-valued factor appear to show stripes, although more breaks (gaps or white space) appear in the stripes as the number of discrete levels increase.

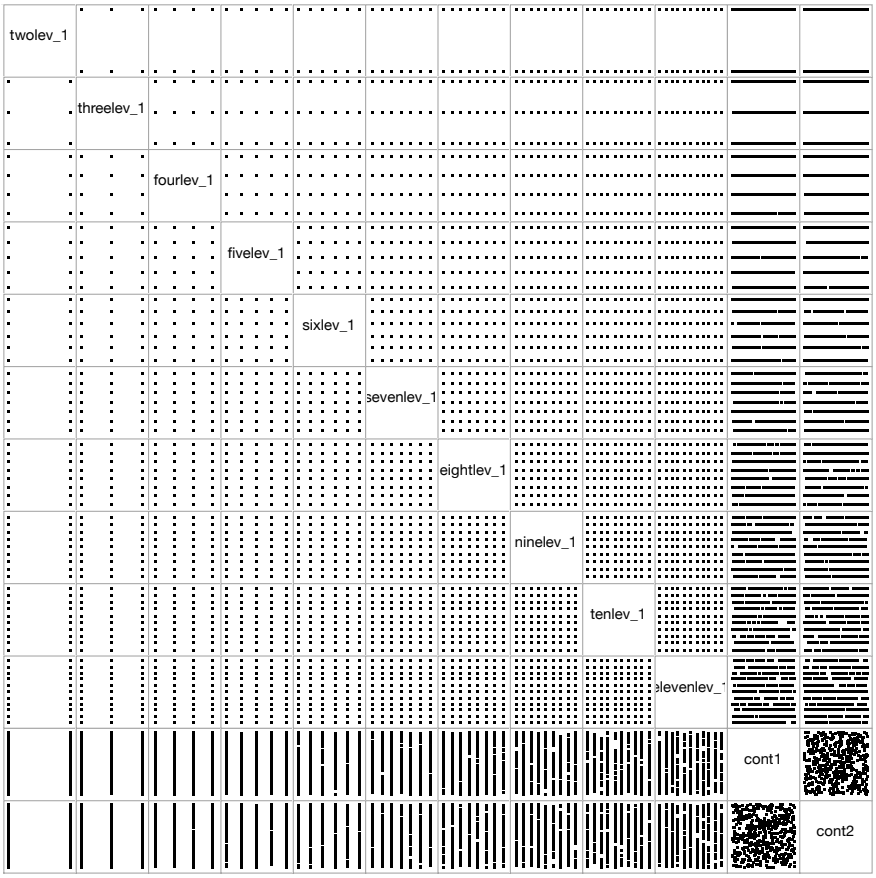


Figure 3.17: Scatterplot Matrix for 12 Factors from the 512-dp NOB.

3.4.4 Frequency-Based Designs

helpful datafarming gem scripts

`generate_design.rb -d fbd` *`options/range-specs`*
generate fully orthogonal second-order designs based on discrete Fourier frequencies, with factor scaling and stacking

Another class of space-filling designs is called *frequency-based designs* (FBDs). These can be constructed by periodically sampling sine/cosine functions at Fourier frequencies ω_i, which are frequencies that complete an integer number of cycles c_i over a fixed sample size n_d:

$$\omega_i = c_i/n_d \qquad \forall\ c_i \in \{1, \ldots, \lfloor (n_d - 1)/2 \rfloor\}.$$

When sines are used as the driving frequencies we generate vectors v_i s.t.

$$v_i = \sin(2\pi\omega_i k) \qquad \text{for } k \in \{0, \ldots, n_d - 1\}.$$

The resulting vectors have the following well-known mathematical properties:

- v_i and v_j are orthogonal $\forall i \neq j$;
- $\sin(\omega)$ and $\cos(\omega)$ are orthogonal $\forall \omega$;
- products of sines/cosines obey the following multiplication identities $\forall\ i, j \in \{1, \ldots, \lfloor (n_d - 1)/2 \rfloor\}$:

$$\begin{aligned}
\cos(\omega_i)\cos(\omega_j) &= (\cos(\omega_i - \omega_j) + \cos(\omega_i + \omega_j))/2 \\
\sin(\omega_i)\sin(\omega_j) &= (\cos(\omega_i - \omega_j) - \cos(\omega_i + \omega_j))/2 \\
\sin(\omega_i)\cos(\omega_j) &= (\sin(\omega_i + \omega_j) + \sin(\omega_i - \omega_j))/2 \\
\cos(\omega_i)\sin(\omega_j) &= (\sin(\omega_i + \omega_j) - \sin(\omega_i - \omega_j))/2;
\end{aligned}$$

- frequencies outside of the range $[0, 1/2]$ are *aliased* by mapping them back into the range using the "sawtooth" function

$$\begin{aligned}
d &= \lfloor \omega/0.5 \rfloor \\
r &= \omega - 0.5|d| \\
\omega_{alias} &= \begin{cases} r & d \text{ even} \\ 0.5 - r & d \text{ odd.} \end{cases}
\end{aligned}$$

A *feasible* FBD for k factors is constructed by choosing a set of integer numerators c and an appropriate value n_d such that all terms of interest in your metamodel are mutually orthogonal based on the properties enumerated above. An *optimal* FBD is a feasible design with the minimum n_d that preserves the mutual orthogonality of all terms in your metamodel.

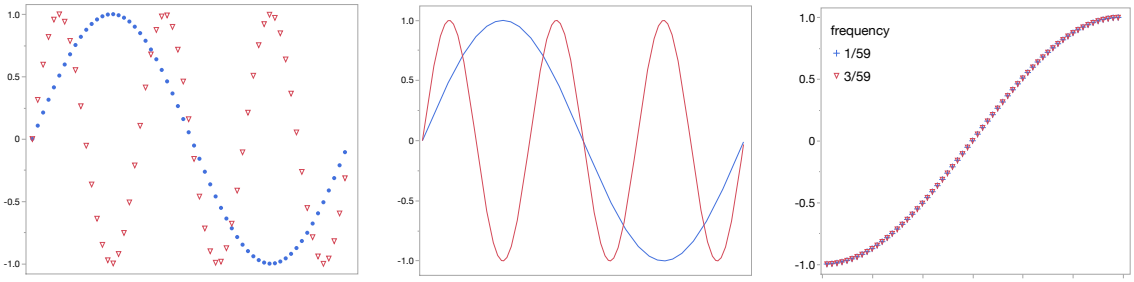
If n_d is relatively prime to all c-values greater than 1 selected as the driving frequency numerators, the resulting design will have n_d distinct values for all factors, i.e., all design points are unique.

Let's construct a second-order FBD for two factors as an example. That means we need to be able to uniquely identify 5 terms in the metamodel: two main effects, two quadratics, and one 2-way interaction. If we drive our two factors using sine functions, both the quadratic terms and the interaction will be observed as cosine terms (via the second multiplication identity given above), and hence will be orthogonal to the main effects even if they manifest at a frequency used for one of the mains. Also note that since $\sin(0) = 0$, by using sine functions for all the main effects the first design point is always the center point of the design space. If we assign frequency numerators 1 and 2, then the quadratic terms will manifest at cosine terms with frequency numerators 2 and 4, and the interaction will occur at cosine terms with frequency numerators $(2 \pm 1) = \{1, 3\}$. The main effects sine term numerators are all unique, as are the resulting cosine quadratic and interaction numerators. Trying different values for n_d, we find that the smallest which does not cause duplication through aliasing is $n_d = 9$. While 9 is not prime, it meets the relatively prime criterion so we're good to go! We can now concisely represent this design as $9 : \{1, 2\}$. The actual design can then be constructed on demand in at most a few seconds using `generate_design.rb`—with scaling and stacking. This is a much more compact way of storing and referencing the design than cataloging an entire $n_d \times k$ matrix, especially for large k (and correspondingly very large n_d).

The design process quickly becomes unwieldy for hand construction, but can be done computationally using an optimization technique called dynamic programming. We provide plots below for two terms with $n_d = 59$ to visually illustrate some characteristics of the designs.

Figure 3.18b shows two continuous sine curves having different frequencies. Figure 3.18a shows the discrete sampling at 59 design points with driving frequencies of 1/59 cycles per observation and 3/59 cycles per observation. Figure 3.18c shows that each of the two factors are sampled at exactly the same set of points; a sufficient condition for this to occur is for the numerators and denominator to be adhere to the relatively prime criterion, and 59 is a prime number. Also note the S-shape of the plot, indicating that design points are denser towards the outside of the range than near the center. This is a positive, as the concentration of points near the boundaries reflects coverage of a greater volume of space in higher dimensions k than LH-based designs with similar n_d achieve.

Earlier work on the development of large-scale second-order FBDs appeared in Sanchez and Sanchez (2019), where the design points were generated based on discrete sampling of cosine (rather than sine) functions, and the design sizes did not include our current relative-primality restrictions for n_d. The current generation of designs, and the corresponding updated software tools, benefit by having fewer design points based on the switch to sines. They also have better space-filling properties by virtue of all design points being distinct.



(a) Sampled values **(b)** Underlying curves **(c)** Sorted sampled values

Figure 3.18: The values obtained by sampling sine curves for driving frequencies of 1/59 cycles per observation and 3/59 cycles per observation appear in (a). The underlying (continuous) sine functions completing exactly 1 and 3 cycles, respectively, appear in (b). In (c), the sampled values that appear as levels in the design matrix are sorted separately for each of the two factors, verifying that these are identical.

Table 3.17 shows design sizes for a variety of k. These second-order FBDs provide several advantages over the second-order NOLH designs:

- FBDs are perfectly orthogonal because of the mathematical properties of Fourier frequencies;
- FBDs are available for much larger values of k; and
- FBDs require far fewer design points for $k \geq 7$.

These FBDs are Resolution V designs so they require larger sample sizes than Resolution III NOLHs, which do not make guarantees on your ability to simultaneously fit nonlinearity and all two-way interactions. It's reasonable to ask how quickly FBDs grow. For a full

Table 3.17: Data requirements for (fully orthogonal) FBDs for selected numbers of factors k, where the design size is prime and the design satisfies the prime/relatively prime criterion.

Number of Factors	Number of Design Points	Number of Factors	Number of Design Points
3	19	15	571
4	37	20	1,409
5	59	30	4,177
6	79	40	8,923
7	119	50	16,229
8	163	60	27,751
9	181	70	42,859
10	251	80	61,223
11	317	90	83,903
12	359	100	109,873

second-order metamodel involving k factors, we potentially need to estimate a β (coefficient) for every term in the metamodel. The full metamodel would be comprised of the intercept, k main effects, k quadratics, and $\binom{k}{2} = k(k-1)/2$ two-way interactions. Consequently, a minimum of $1 + 2k + k^2/2 - k/2$ design points are needed. The two-way interaction term is dominant, yielding a lower bound of $\Omega(k^2)$ for the growth rate. In practice, empirical evidence indicates that the actual growth rate is greater than k^2 but less than k^3 (Sanchez and Sanchez, 2019).

Figure 3.19 shows the scatterplot matrix for 4 stacks of an FBD for $k = 10$ factors, where the first stack is highlighted in dark blue and the others are light gray. Visually, these provide dense coverage in two dimensions with higher density near the edges as discussed previously.

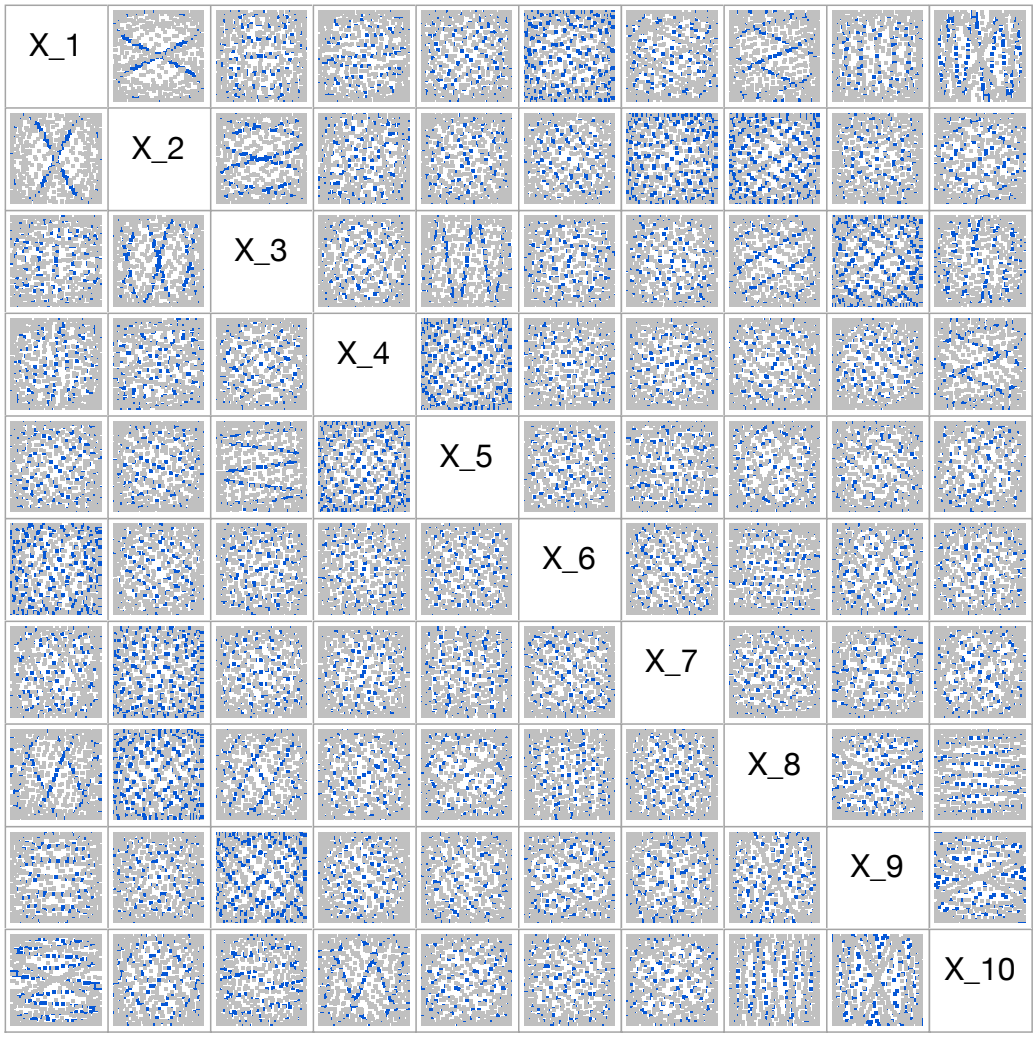


Figure 3.19: Scatterplot Matrix for 4 stacks of an FBD for $k = 10$ factors.

3.4.5 Frequency-Based Screening Designs

helpful datafarming gem scripts

`generate_design.rb -d fbs` *`options/range-specs`*
generate orthogonal, second-order identifiable designs based on discrete Fourier frequencies, with factor scaling and stacking

Another class of designs, which we call *frequency-based screening* (FBS) designs, is a variant of FBDs that is more efficient but not completely second-order orthogonal. FBS designs are, however, capable of fitting full second-order metamodels—something the NOLHs and NOBs cannot do. We will say these are *second-order identifiable* rather than second-order orthogonal. We construct these so that main effects are orthogonal to all quadratic effects and all two-way interactions, but allow for some correlation between different two-way interactions or between quadratic effects and two-way interactions in the the analysis matrix.

3.5 Implementing the design

helpful datafarming gem scripts

`rundesign_general.rb` *`options`* `‘cmd‘ design_file #reps output_file`
run control to apply a designed experiment to a model with replication

Once you've selected a design you are ready to grow your data. No matter which design you choose, you will need appropriate run control to generate the data for subsequent analysis.

3.5.1 Run control

The `rundesign_general.rb` command supplied with the datafarming Ruby gem allows you to run any simulation that can be executed from the command line and controlled by command line arguments, such as the reference simulations accompanying this book. Type

```
rundesign_general.rb -h
```

on a command line for help and more details on the options. This script takes four inputs: the command to execute (in single quotes), the design file (text-delimited, with no headers), the number of replications, and the name of an output file. For example, the `epidemic.rb` simulation takes five command-line inputs: the population size, number initially infected, transmission ratio, intervention day, and the transmission ratio reduction (as a proportion)

after the intervention. We could generate an FBD and get the results of 50 replications in the file `myout.csv` by typing

```
generate_design.rb -d fbd --no-headers id:pop,min:4000,max:8000,decimals:0
       id:init,min:1,max:10,decimals:0 id:R0,min:1.25,max:3.5
       id:day,min:1,max:30 id:reduc,min:0.3,max:0.9 > mydesign.txt
csv2blank.rb mydesign.txt
rundesign_general.rb ‘ruby epidemic.rb‘ mydesign.txt 50 myout.csv
```

Alternatively, if we've generated a design with headers, we can remove those by typing

```
stripheaders.rb mydesign.txt
```

before calling `rundesign_general.rb`.

Note that `rundesign_general.rb` can also be used to automate the execution of experiment runs for simulations written in Python, Java, R, or other languages. Other run control options are also possible. For example, the SimOpt testbed is open source and freely available at `https://github.com/simopt-admin/simopt`. It permits you to run data farming experiments involving parameters of simulation optimization solvers, as well as factors for the simulation testbed models. Data farmers have also written custom "data farming wrappers" for many other simulation platforms over the years. Sometimes these are needed to be able to run the experiment in batch mode, rather than make changes manually using a graphical user interface. Sometimes the simulation may have a built-in "scenario" or "experiments" page (as is the case for Simio releases 17 and later). Alternatively, there are other run control environments that have been developed, such as SESSL (Ewald and Uhrmacher, 2014). Ultimately, it is critical that the the factors and outcomes be placed in a one-to-one correspondence in a file (such as a .csv file) that can be imported to a statistics package for further analysis. Your simulation should echo its inputs alongside the outcomes, or you will need to find some mechanism (or wrapper) to collate the two components after the fact.

3.5.2 Leveraging cores, clusters, and clouds

If you are working on a multi-core machine, or have access to cluster or cloud computing, the run execution can be sped up significantly by using suitable cluster control software. You will need to invest some time to learn how run control is handled—a task that is outside the scope of this book. With modern cluster infrastructure the speedup is nearly linear in the number of cores available. We have had excellent results using HTCondor (HTCondor.org, 2025), which is an open source software framework freely available from the Computer Science department at the University of Wisconsin.

Fortunately, the run execution is very easily parallelizable, as each run can be launched separately without the need for parallelizing within the simulation itself. This tends to balance the load across multiple cores despite potentially high differences in processing times for different runs.

3.6 Data farming considerations

Overall, keep the "think big!" viewpoint in mind as you prepare your data farming experiments. Think big in terms of the factors to explore, the features of the response surface, and the flexibility you will have for subsequent analysis.

3.6.1 Choose factors wisely

In physical experiments, it's best to think carefully about a specific question (or two) you'd like to answer, and set up the experiment specifically to address this question. Because control is difficult, you may have to limit yourself to a very small number of factors. For instance, in the gummy bear experiment discussed in Table 3.1 we focus on only two: launch position and launch angle. You should also make use of blocking and randomization. Use blocking if there are other aspects that aren't specifically of interest, but could otherwise creep in and affect the results. Blocking can also provide greater power of hypothesis tests and precision in estimates. Randomization guards against unknown sources of bias.

We take a much broader view when performing simulation experiments. Generating a list of the potential inputs to a simulation model is one way of coming up with an initial factor list. However, it is important to remember that factors need not correspond directly to simulation inputs. Potential factors include the *input parameters* or *distributional parameters* of a simulation model. For example, suppose two inputs are the mean times μ_1 and μ_2 required for a specific agent to process messages from class 1 and class 2, respectively, where message class 1 is considered more complex than message class 2. Varying μ_1 and μ_2 independently may either result in unrealistic situations where $\mu_1 < \mu_2$, or require the analyst to select very narrow factor ranges. Instead, we could use μ_1 as one factor to represent the capabilities of the agent, and vary the ratio μ_2/μ_1 over some range of interesting values (say, 0.4 to 0.9) to represent the relative difference in message complexity.

Factors can even include modeling choices such as which distributions to use. This can be particularly useful in the *farming for modeling* phase of a study, where we may not yet have had the opportunity to collect much if any data. This approach can help identify how much impact alternative modeling choices would have on the system behavior. If the impact is minimal, then this is an opportunity to avoid unnecessary and potentially costly and time-consuming data collection and modeling effort, avoiding the excruciating level of detail described in the pizza model of Section 2.1.

On the other hand, sometimes a more abstract view of a factor is more appropriate. For example, in an agent-based version of the capture-the-flag game, where (say) 10 players are on our team, there are many ways of incorporating speed and stealth.

- All team members have the same speed and stealth, in which case we have just two factors. If our simulation requires separate inputs for each individual's speed and stealth levels, the model has 20 inputs. By lockstepping all the speeds, and similarly

lockstepping all the stealth levels, we can generate the 20 required inputs from two design columns.

- Each agent has individually specified speed and stealth, in which case our design requires 20 factors. One issue with this approach is that k is based on the number of agents. A more important issue relates to the interpretability of the results. For example, if all agents are interchangeable in terms of the program logic but only agent 3's speed turns out to be important, what does that mean to the team's coach?
- Each team member has a unique speed and stealth, but these are randomly drawn from two input distributions at the beginning of each run. For example, suppose we use uniform distributions $U[a_1, a_1 + b_1]$ for speed and $U[a_2, a_2 + b_2]$ for stealth. We could then include four factors in our experiment, regardless of the number of players. The first pair of factors a_1 and a_2 are the minimum speed and stealth, respectively. The second pair of factors b_1 and b_2 characterize the variability in our players' abilities. Factors could represent other parameters of the distributions. For example, we could vary the averages and standard deviations of the uniform distributions, rather than the minima and ranges. We could even vary the choice of distribution used in the model.

The third approach has several benefits. It facilitates efficient experiments involving large numbers of agents. Interpretation is straightforward—we will find out the impact of overall speed and stealth, as well as heterogeneity among players, on the game results. And we can evaluate the impact of alternative model-building choices.

3.6.2 Choose a reasonable experiment size

The time required to run the experiment will depend on the execution time of the simulation (which can vary greatly across dps or replications), the number of replications, the number of design points, and the degree of parallelization. Before performing the "production" runs of a large-scale experiment, it is often helpful to get a ball-park idea of the amount of time that will be required. To do this, you can start by making some test runs with timings to observe the amount of time required for a single replication of a single dp. If some dps are likely to take more time than others, you can either focus on a few dps that are likely to have long run times to get an approximate upper bound, or a single replication of a (possibly smaller) design to get an idea of the average runtime. These timing operations do not require sophisticated statistical tests, they just provide practical guidance for how long you have to wait before you get to the analysis, a.k.a. the fun part.

Let's consider a numerical example to illustrate what drives the experiment size decision. Suppose you have a model with 20 factors that you want to explore. A 2-level ResV fractional factorial has 512 dps, and if you wanted 5 replications that would require 2560 runs of the model. In contrast, a 20 factor FBD has 1409 dps, and would require 7045 runs to do 5 replications. These are both numbers that historically would have been terrifying to analysts running physical experiments, given the investment in time, labor, and resources, as well as

the high likelihood of human error in manually collecting, collating, and performing data entry. However, with computer experimentation we can benefit greatly from automation of run control and parallelization. If you find via benchmarking that the average runtime for your model is one minute, then we would project that running the experiments sequentially with the ResV design would take 2560 minutes, or 2560 / 60 = 42.7 hours, while for the FBD it would take 7045 minutes or 7045 / 60 = 117.4 hours. Our current laptops have 8 cores, and if we assigned 5 of those to doing the runs we would still have a comfortable margin to do other work. The overall times would be reduced to a bit more than 8 hours and a bit less than 1 day for the ResV and FBD, respectively. Given access to a cluster, those times would both be reduced by an order of magnitude. Based on our earlier discussions about the benefits of space-filling designs, our personal preference would be to use the FBD since it allows far more flexibility in constructing metamodels and far better lack-of-fit detection. In either case, we sidestep the issues of collecting, collating, and transcribing the data by using automation. Obviously the actual times would scale with model run times.

The overarching goal is to choose a design, run length (for non-terminating simulations), and number of replications that meets your current needs. At one extreme, in the *farming for modeling* loop when you are most interested in model verification (debugging) during the model building process, instant or near-instant turn-around is important. Consequently, you may be satisfied with a few replications of a very small design or even a small set of dps chosen from a larger design. At the other extreme, after going back and forth through the *farming for modeling* and *farming for insights* loops multiple times, you may end up determining that a large design with multiple replications that takes a few weeks of cluster computing time is appropriate for growing the data needed to move to the *insights to impact* loop.

Terminating simulations

For terminating simulations, this means choosing a suitable number of replications, since each run is an experimental unit. The number of replications depends primarily on the types of responses. If all your responses are continuous-valued, then even a handful may be sufficient for identifying the most impactful factors via metamodels, as we will discuss in Chapter 4. However, if any of your responses are binary, the number of replications may be much larger—particularly when simulating rare events. If, say, you are interested in how well different power grid configurations avoid unplanned shutdowns of 12 hours or more, and you anticipate the probabilities of such events to be in the range of 0.0–0.001 in a single run, you should be making thousands or tens of thousands of replications to detect differences across the dps.

The number of dps depends, of course, on the design you choose—including the number of stacks. What is key in data farming is to focus on practical issues, not on trying to minimize the total time. In simulation experiments, where a major portion of the effort often occurs in model development, the total sampling budget may not be so constrained. This increases

the set of potential designs that can be used, and it may be possible to generate a great deal of information (even hundreds of thousands of runs) in a relatively short time. This makes it possible to "think big"—in fact, we can think of simulation experiments as providing better data, not just big data, as discussed in Sanchez and Sanchez (2017) and Sanchez (2018). For example, if you are making runs on a single machine, it may be perfectly acceptable for your experiment to take 12 hours to run rather than 2 hours to run—particularly if you start the runs at the end of a workday and don't plan to start analysis until the morning. Similarly, if you have access to cluster or cloud computing with 2000 cores, then arbitrarily restricting yourself to a handful of replications of a 40-dp design D is wasteful of your own time. Other options with roughly equivalent clock time include 50 replications of D, 25 replications of two stacks of D, or 10 replications of five stacks of D. Any of these will improve your ability to glean insights from the data. Stacking the design will improve its space-filling properties and can provide much more detailed insights into your simulation model's behavior.

Non-terminating simulations

Non-terminating simulations model real-world systems which lack a clearly defined stopping condition. For terminating simulations, there is no additional information to be gained once the terminating condition is fulfilled. For non-terminating simulations, running the model for a longer period of time yields more information when the following working assumptions are met:

- The system is ergodic, so all states are reachable from any other state, but state transitions are not cyclic. Counterexamples of this assumption would include systems with "absorbing" states or subspaces, or systems which cycle through the entirety of their state space in a deterministic order.
- The system is stable and will eventually settle into some "steady-state" mode of behavior, where the expected value and variance/covariance structure of the output are no longer dependent on the time index. The technical term for this is *covariance stationarity*.

The `fleet_availibility.rb` simulation is an example of a system with this type of behavior.

Systems which are covariance stationary can be analyzed using a variety of techniques. Counterexamples which range from hard to impossible to analyze would include non-stationary systems with explicitly time-dependent variations in the mean or variance/covariance structure of the output. For instance, systems with deterministic trend lines or cycles in the output data cannot directly be evaluated using techniques we describe below. The data farming analyst needs to be aware that poorly chosen or wide parameter ranges can cause non-stationarity at some design points, and it is important to modify your design or responses accordingly in order to enable valid analysis of the results. We discuss this further in Section 5.11. but we will assume that we are dealing with covariance stationary data for the remainder of this section. We will focus on replication/deletion and batch means as two analysis techniques that are relatively straightforward to understand and use.

A simulation model is a computer program, so like all programs the simulation model's variables require initialization. In particular, we must choose a starting state for our model of the system. We seldom know the steady-state characteristics of the systems we model using simulation, so the program is usually initialized to a convenient state, i.e., a state that's easy to program but which has a long-term low probability of occurrence. For example, queueing systems are often started with all servers idle and all queues empty. Performance statistics that include data collected while the simulation is "warming up" from an atypical initial state will not be representative of their steady-state measures, i.e., they will be biased. The bias diminishes asymptotically, but in practice we want reasonably unbiased answers quickly. A common approach is to try to assess how long the warm up period lasts, and truncate the initial transient to remove or at least mitigate the bias. This technique is common to both replication/deletion and batch means.

Replication/deletion revolves around making multiple independent runs of the simulation at each design point. After identifying and removing the effects of initialization bias from each run, aggregate statistics are calculated based on the remaining data for that run. Since the runs are independent, we can then use classical statistics to calculate means and variances of the (hopefully unbiased) i.i.d. aggregate data at each design point. The "pro" for replication/deletion is that the estimates are based on truly independent data if the replications are performed independently. The "cons" are that (i) data is truncated from every run, so degrees of freedom at each design point come at a cost of throwing away more data; and (ii) there's a temptation to control the budget and experimentation time by making shorter runs, but short runs create a greater likelihood of mis-estimating the required truncation amount, resulting in biased estimates of the steady-state behavior.

To use batch means estimation, we make a single very long run at each design point. After identifying and removing the effects of initialization bias, the remaining data is broken into batches, and aggregate statistics are calculated for each batch. If the batches are sufficiently large and non-overlapping, they are considered to be nearly independent and we then use classical statistics to calculate means and variances of the (hopefully unbiased) i.i.d. aggregate data at each design point. A more sophisticated technique developed by Meketon and Schmeiser (1984) uses overlapping batches and a careful weighting scheme to cancel the correlation from the overlaps. In order to get equivalent degrees of freedom to replication/deletion, we make a single run that is many times longer than any of the replication method's runs. This is a "pro," because (i) we are better able to estimate the correct amount of data to truncate; and (ii) there is only one truncation action required per design point so batch means is less wasteful of data. The "con" is that our estimates are based on data that are "mostly," but not completely, independent.

In the real world we need to set limits on the combination of run lengths and number of runs we can perform. The analyst must consider trade-offs between several aspects of their analysis, including the number of factors, the corresponding design requirements, and the degrees of freedom needed. Degrees of freedom for non-terminating simulations are determined by the number of replications or batches. This leads directly to another choice you must make

as an analyst, the stopping condition. This might be either a specific time—e.g., stop after 1,000 simulated days have elapsed—or a specific number of output observations—e.g., stop after 100,000 customers have been served. As discussed above, each run will likely require a warm-up period.

At the planning stage, consider that you need fewer replications if you have longer runs. With replication/deletion, longer runs yield smaller standard errors, so fewer runs are required to achieve a given level of precision for estimators at that design point. Conversely, a batch means approach involves only one very long run at each dp. If there are multiple performance measures, it makes sense to use the same truncation point so that all are based on comparable data. For ease of implementation and analysis, we can extend this to using a common truncation point for all production runs. Now we just need to identify it!

The datafarming Ruby gem provides `mser.rb`, a script that can be used to determine a suitable truncation point for a single stream of time-series data. It is based on the premise that initial bias occurs at the front end of a time series, and the data least likely to be affected is at the tail end of the time series. A MSER (Mean-Squared-Error-Reduction) estimator is calculated for the last half of the time series data, then updating the estimator by adding observations from the midpoint to the beginning. Using the second half of the data yields a yardstick for evaluating the first half. MSE should decrease with additional observations unless there is a shift in the underlying distribution, so the location of the minimum MSER estimator when moving backwards is deemed to be the point at which the time series transitioned from biased to unbiased. See Sanchez and White (2011) for more information on the method.

You may want to run `mser.rb` on either a single replication of the entire design or on a few test runs of dps that you anticipate may have long warm-up periods if those are known *a priori*. Rounding up to a larger truncation point provides an additional guard against mistakenly retaining part of a warm-up period in later runs, and making this a nice round number (say, 1000 instead of 823) reduces the need to go into details of the method. For example, below are results from 8 replications of a dp of an early variant of the fleet availability simulation where the fleet was brought up to full capacity over a fixed period of time. The three columns are the average number available, the number of usable points, and the number to be truncated for each of the replications.

```
x-bar,n,trunc
24.881443,194,6
29.319149,188,12
26.514706,136,64
29.294118,187,13
27.549738,191,9
28.101266,158,42
30.176871,147,53
29.188482,191,9
```

From these, we see a lot of variability in the truncation point, which ranges from 6 to 64. We might decide to round the truncation point up to 100 going forward. We might also decide to increase the run lengths from 200 days to a larger number so we truncate a smaller proportion of the production run data. This may have a noticeable impact on the total runtime. For example, suppose from our test runs, our ball-park estimate of total runtime for 10 replications of design D is 10 hours if each run length is 200 days and we truncate the first 100. If we wished to cut all standard error estimates in half, we need roughly four times as much usable data. This could be achieved by conducting 40 replications instead of 10 (for an estimated runtime of 40 hours), or by making each run of length 500 instead of 200 (keeping 400 days of data from each run for an estimated runtime of 25 hours). Also consider whether adding stacks or using cluster or cloud computing would affect the run time or clock time.

We remark that warm-up periods can be quite large. In queueing systems with high traffic intensity, it is common to need truncation points of several thousand departures.

If you consider batch means to be an attractive option, the datafarming gem provides two analysis tools as options: `mser_nolbm.rb` and `mser_olbm.rb`. Both use the MSER technique to determine a suitable truncation point, and based on that determine how many batches can be created using the remaining data. The two scripts then calculate the batch means statistics using non-overlapping or overlapping techniques, respectively.

3.6.3 Ignore or exploit sequential run control

Depending on your simulation's run time, you might or might not be interested in manipulating the order in which the runs are executed. This can be done either by using the fixed-size designs discussed earlier in this chapter, or by using a sequential design approach.

Sequential sampling for fixed-size designs

A fixed-size design refers to any of the designs discussed earlier in this chapter where the design is specified before running the experiment. Two basic approaches are: (i) complete a full replication of your design before moving to the next replication, and (ii) complete all replications at one dp before moving to the next dp.

The first is beneficial if you may want to peek at your results before the entire experiment is complete, and is how `rundesign_general.rb` operates. This may be of particular interest if a single run of your simulation takes a long time; if you are still in the *farming for modeling* phase; or if there is some chance that you may be asked for a preliminary analysis before all replications have been completed. For simulations that do not include the number of replications as an input, such `gummies.rb`, this happens automatically if you use `rundesign_general.rb` as the run control mechanism. If you stop halfway through the time required to run the entire experiment on a single processer, you will still have data from

several complete replications of the design. In the extreme, you could generate the data one replication at a time, and conduct analysis after each one to determine whether or not additional replications are warranted.

For simulations that include the number of replications as one of the inputs, you have options. For example, `epidemic.rb` has 6 inputs, the first being the number of replications. You can execute 50 replications in several ways, including setting `n_reps` to 1 in the design file and set `n_reps` to 50 when invoking `rundesign_general.rb`, if you want to build the output file one replication at a time. If your simulation takes a very long time to run, you could conduct preliminary analyses after each replication is complete, and stop growing data as soon as you decide you've seen enough. If instead you set `n_reps` to 50 in the design file and set `n_reps` to 1 when invoking `rundesign_general.rb`, you may not have data from each dp if you stop or take a peek at the data partway through. If you stop halfway through the time required for the entire experiment, you are likely to have data missing for a sizable proportion of the dps. How much data is missing depends on the number of dps that require above-average runtimes, the number of cores available, and the order in which the dps are processed.

Sequential sampling can also be exploited by expanding the design over time. Suppose you begin with a single stack of design D. If, after preliminary analysis, you wish to see more detailed results, you can augment this design with additional stacks or a different design with the same factor ranges. For example, you could add an FBD to a rotatable CCD to get a more space-filling design, or add an FBD to an NOLH (or vice versa) since, when k is large, the former samples more densely near the edges of the factor input region and the latter samples more densely near the middle. Another way to expand the design involves crossing. Suppose you experiment with a simulation of a hospital emergency room. Your initial experiment D involves factors such as the numbers of beds, doctors, nurses, etc., and uses a first-come, first-served rule for patients entering the waiting room. After this experiment is complete, rerunning D with one or more new priority rules is, in fact, using a crossed design $X \times D$ where X is a qualitative factor with priority rules as the levels.

Sequential and adaptive sequential design approaches

Although not the focus of this text, recent work on sequential and adaptive sequential designs is promising for future data farming studies. Sequential designs refer to designs that do not have a fixed sample size but evolve over time. Two sequential approaches that improve their space-filling behavior over time are Sobel sequences Sobol (1967) and the sliced designs of Duan et al. (2017). The latter takes designs in slices of size k for continuous valued factors. At certain stages, the designs can be considered either Latin hypercubes or full factorials when the points are viewed on a grid. Note that these sequential design approaches do not make use of any information about the responses. Those that do are called adaptive sequential designs.

3.6.4 Leverage reproducibility

If your simulation has the option of user-specified random number seeds, then the results are reproducible. This can be useful for several reasons. First, it facilitates debugging during the simulation model development. The ability to reproduce error-generating data is essential to debugging. For example, suppose in looking over results from a logistics delivery system you notice that occasionally a delivery vehicle teleports to the next location. Rerunning that scenario after updating the simulation logic will reveal whether or not you your code fix actually worked as intended.

Reproducibility is also useful because it can permit flexibility in the type and amount of output generated. For example, the `fleet_availability.rb` simulation has a toggle allowing you to specify whether you want daily availability reports or you want summary measures (mean, standard deviation, and quantile information) at the end of the run. The summary data requires much less disk space, and can also affect the runtime for the experiment as a whole. We have worked with simulations where the time to complete the detailed runs is an order of magnitude higher than the time required when only summary output is reported. We have also worked with simulations where a single replication can generate tens or hundreds of GBs of output data Morgan et al. (2018). In such cases, not only is the I/O process slow, but bandwidth limitations can bog down the I/O transfer needed to consolidate output data from multiple cores of a computing cluster or cloud.

Reproducibility is also helpful for simulations that have optional animation features. The animation should be turned off when you make production runs. However, any run yielding interesting or surprising output can be rerun with animation, which may reveal some insights not easily gleaned from numerical summaries alone. For example, consider an agent-based capture-the-flag simulation where the results for a speedy, stealthy team were extremely bad. If a visual inspection of these indicated that the teammates stayed very close to each other in bad runs, while they tended to disperse quickly in good runs, that might merit further exploration. Metrics about team dispersion may either already be available from detailed output or quickly obtained from post-processing this output. If so, this apparent finding can be assessed across the entire experiment to see whether or not it holds up. If not, the code can be modified to include this as an output. If it turns out to be important, this might suggest another coaching opportunity. The modeling team might respond by revising the movement algorithm, for instance by putting some weight to make team members seek separation from each other unless they see the opposing team's flag.

Chapter 4

Metamodeling

Key Concepts

- Metamodeling is a powerful way to summarize relationships, including the cause-and-effect relationships between the factors and responses in datasets generated via designed experiments.
- Metamodeling gives you three things about a relationship: a mathematical expression, a characterization of the uncertainty present, and statistics that indicate its strength.
- Graphical techniques can reveal patterns and aid in metamodel selection and interpretation.

The term *metamodel* means a model of a model. This is a commonly used term in simulation. If a simulation is an attempt to model reality, then a statistical model (such as a multiple regression) of the simulation output is a metamodel. After a brief section on notation and definitions in Section 4.1, we discuss models and metamodels in Section 4.2. Statistical metamodeling (in its various forms) is extremely valuable. If we think of statistics as the process of turning (potentially very long) lists of data into insight, then metamodeling is a way of summarizing relationships. Spell-checking programs may want you to add a hyphen (i.e., "meta-model"), but the standard usage in the simulation community is a single word, so that's the convention we'll use. For each response Y, metamodeling provides:

- a mathematical expression of the relationship between the factors and the response,
- a characterization of the uncertainty present in this relationship, and
- statistics that indicate the strength and significance of the relationship.

A variety of metamodeling and visualization approaches have been successfully used for data farming. In this chapter, we focus on two primary types: *multiple regression metamodels*

in Section 4.3; and *partition tree metamodels* in Section 4.4. The former are parametric models that are well-suited to characterizing relationships for many types of smooth response surfaces, while the latter are nonparametric models that are particularly good for response surfaces that either have abrupt changes or discontinuities, or involve qualitative responses. Partition trees can also be translated into a multiple regression context, making it possible to leverage the strengths of both techniques by combining them to form a hybrid model, as described in Section 4.5. Other types of metamodels that have been used are briefly described in Section 4.7, with references to sources that provide more detail. Readers familiar with this early material and interested primarily in a practical guide to metamodeling for data farming may choose to focus on Section 4.8.

Bearing in mind that the human eye is a powerful tool for pattern recognition, graphical techniques can be very helpful. Large tables of numeric data can become overwhelming, but graphs and plots often reveal patterns or problems that are difficult to uncover by numerical summaries alone. In this chapter, we provide examples of graphs that are useful for fitting, assessing the quality of, or interpreting metamodels. More information on visualization techniques that are appropriate for data farming appear in Chapter 5.

4.1 Notation and Definitions

Throughout this chapter we will use Y to denote the *response* of interest. Commonly used alternatives are *dependent variable*, *measure of effectiveness* (MOE), or *measure of performance* (MOP).

We will let X denote an *independent variable* or potential *explanatory variable*. If we have k potential explanatory variables, we use subscripts to denote vectors of them as $X_1, X_2, \ldots, X_k$. Data farming studies typically involve $k > 1$, in which case we describe the response as a *response surface*. Many data farming studies performed by the NPS SEED Center have involved dozens of factors, and in some cases we've dealt with k as large as several hundred or even a few thousand.

As in Chapter 3, we will refer to the Xs that we vary in purposeful ways as "factors" rather than "variables." The software used to construct metamodels doesn't care whether or not data come from a designed experiment, nor whether the data are collected in the real world or generated from a simulation. However, the fact that the data are generated in a smart way has a direct impact on what types of metamodels can be fit, and makes life easier for analysts and decision makers by avoiding many of the fitting and interpretation issues that can arise when dealing with observational data.

4.2 Philosophical Interlude: Models and Metamodels

In its most general form, we can think of characterizing the relationship between a vector of simulation output responses Y and a corresponding set of vectors of simulation inputs $X \equiv \{X_1, \ldots, X_k\}$ as

$$E[Y] = g(X) \qquad \text{and} \qquad Var[Y] = h(X) \tag{4.1}$$

where $E[Y]$ and $Var(Y)$ are the expected value and variance of the response, and the functions $g(X)$ and $h(X)$ are the corresponding true (but unknown) mapping relationships.

One of the most basic parametric forms of metamodeling (and one we assume you are already familiar with) is *simple regression*, which involves a single X. The underlying regression model has the form:

$$Y = \beta_0 + \beta_1 x + \varepsilon, \tag{4.2}$$

where Y is the response corresponding to the input value x, the βs are deterministic coefficients, and ε (often referred to as *error*) is a random variable with mean zero and constant variance σ^2 that is the source of i.i.d. variability in the Ys. The ε is often assumed to have a normal distribution—an assumption that is not needed to estimate the βs using least squares, but enables probabilistic statements about the resulting regression model. We will revisit all these assumptions shortly.

Taking the expectation of both sides of equation 4.2 provides a fairly common alternate representation for the relationship:

$$E[Y|x] = \beta_0 + \beta_1 x. \tag{4.3}$$

Alternate notations include $E[Y_x]$, $\mu_{y,x}$, or simply y. The least squares method is used to estimate the various coefficients from a set of data $\{x_i, y_i\}$. The resulting *predictive model* or *metamodel* is written as

$$\hat{y}_x = \hat{\beta}_0 + \hat{\beta}_1 x \tag{4.4}$$

where the "hats" on β_0 and β_1 indicate these are estimated from data, and the "hat" on y_x denotes that it is a point prediction of the expected value of Y based on the estimated βs. Note the correspondence with equation 4.3.

Several other statistical measures are commonly reported. The first is an estimate of either the error variance σ^2 or error standard deviation σ, which are called the *mean square error* (MSE) and *root mean squared error* (RMSE), respectively. The second is an estimate of the *statistical significance* of the regression, typically a p-value associated with the null hypothesis that the slope β_1 is zero. Low p-values indicate low probabilities of concluding a relationship exists due to spurious correlation. The third is an indication of the *practical strength* of the linear relationship in terms of the correlation coefficient r. Recall that $-1 \leq r \leq 1$, and that $r = \pm 1$ if and only if x and y follow a perfectly linear relationship with no uncertainty.

Equation 4.2 is often presented as the "ground truth" for a regression problem, where the βs and σ^2 are described as the true, but unknown, coefficients of the underlying relationship. Equation 4.4 is then a statistical model of that relationship. But let's look at this a little more closely with the view that equation 4.2 is, in fact, a model of reality.

Consider a simple physical experiment where you're giving subjects a number of tasks to complete (x, ranging from 1 to 10) and recording the total amount of time it takes for them to complete these tasks Y (in seconds). Let's discuss the assumptions we'd be making.

- **Linearity?** At first glance, this may seem like a no-brainer. If it takes 42 seconds to accomplish one task, shouldn't it take 420 seconds (or seven minutes) to accomplish ten tasks? Why might things go wrong?
  - **Learning curve.** Suppose the task is "play a new game N_{task} times" or "use a new software interface to perform N_{task} operations." When a subject walks through the door, they're told how many tasks they must do—that is, they'll find out their personal number of N_{task}—and have a single value recorded corresponding to the total task completion time. The first few times through, people may require more time because they're still learning the rules or getting familiar with the system. Consequently, a person doing many tasks is more likely to spend less time per task, on average, then someone who's just doing one.
  - **Fatigue**. Suppose the task is "run a mile." Most serious runners would pace themselves differently (and go slower, on average) if they ran ten miles instead of one mile. Casual runners might slow down tremendously, or even switch to a combination of running and walking, at longer distances.
  - **Setup time**. Suppose the task is "pour a cup of coffee." There might be some time required to pick up the coffee pot that would be added only once, whether someone poured one cup or ten cups. This is not actually a problem for a regression model—it simply means we have a non-zero intercept. However, it may be different than how a person on the street would think of linearity, in terms of a straight multiplication of the time.
- **Independent errors?** The assumption of independent errors means that knowing something about one of the errors doesn't give any useful information about the others.
  - **Environment changes over time.** Suppose the task requires inserting two components together, and it's a little harder to do under low lighting conditions. If you had recorded the time at which the observations were taken, and noticed a difference between those during the day and at dusk, this might jump out at you.
  - **Repeated measures.** This occurs when multiple observations are taken from subjects who have different underlying means. Suppose Subject A comes in, and you have them run through several potential task numbers in random order (e.g., first they are timed for completing three tasks, then for seven tasks, then for one task, etc.). Subject B comes in and does something similar (e.g., they first

complete all ten tasks, then two tasks, then four tasks, etc.). If Subject A happens to be fast and Subject B happens to be slow, then the errors aren't independent if you fit a model without including the Subject information. This might or might not be evident from the scatterplot of residuals versus predicted values, depending on how different the two subjects are and how much data you've gathered, unless you use color codes or distinct markings for the two different groups.

  - **Hidden reasons.** These might or might not yield outliers. Classifying something as an "outlier" when we go to fit a metamodel means that it's unusual data. One reason an outlier might occur in practice is that there is something fundamentally different about the circumstances that gave rise to a specific data point. If you thought your subjects were all comparable and that environmental conditions, etc., would not have any affect on the results, but later found that your unusual observation was from a faculty member and the rest were from students (or the unusual one occurred during a fire drill and the rest did not), that might make you think about why the results were different. This might end up being the most interesting part of your data! Hidden reasons such as these may induce correlations even if they do not result in outliers.

- **Normally distributed errors?** The normal distribution is a favorite of statisticians because it often arises in practice, such as when adding up lots of smaller random elements. It can also be a useful model for measurement errors—if you remember your basic chemistry lab, you might have been asked to weigh every sample three times. That's because sometimes the scale might read low, and sometimes high—but we expect these errors to be fairly symmetric and usually small, and a bell-shaped curve seems reasonable. However, it's not the most reasonable assumption for all situations. A few counter-indications are listed below.
  - **Truncated data.** A normal distribution theoretically can take values anywhere from $-\infty$ to ∞, although 99.73% should fall within the range $(\mu \pm 3\sigma)$. If we assume the errors have mean zero, that means that most of the data will fall inside the range $(\pm 3\sigma)$. But if we're measuring something like elapsed time, we know it can't possibly be a negative number. Also, if you're collecting data and (say) someone runs from the room with flu symptoms before they finish their tasks, you wouldn't keep waiting for them to return so you could record their total time. Ignoring the truncation issue will not be a problem if the errors aren't likely to run into either the high or low ends, but this assumption isn't appropriate if the "hard" limits affect the data.
  - **Skewed data**. If you've watched the CBS show "Amazing Race," you know that contestants are sometimes given a task like "put flags on a flagpole in the order of the countries you visited." The task is not complete until they are successful. If contestants do this right the first time, the time is based on how long it takes to attach the flags and hoist them up the pole. However, if they make a mistake and have to take them down and try again, it can take much longer. The underlying

distribution may be skewed rather than symmetric. This is common in truncated data where the truncation is one-sided.

- **Constant variance?** We know this doesn't hold for a lot of systems, even if we look at basic probability principles. For instance:
  - **Learning effect applies to variability.** If a subject doing ten tasks is likely to get into a rhythm after a few, there might be less variability around the time to complete ten tasks than around the time to complete only one.
  - **Observations from subjects have different underlying variances.** Similar to what was described above, suppose you have two (or more) groups of subjects, and the task is to "find the peg that fits in a hole." Those in Group A are very methodical and consistent, while those in Group B approach things haphazardly and might be exceptionally lucky or exceptionally unlucky. This might or might not be evident from the scatterplot of residuals versus predicted values, depending on how different the two groups are and how much data you've gathered, unless you use color codes or distinct markings for the different groups.
  - **Proportional errors**, also called multiplicative errors. For many systems, the variability is related to the mean. For example, if you flip a fair coin once, the number of heads is either 0 or 1, the expected number of heads is $\mu = np = 0.5$ and the variance is $\sigma = np(1-p) = (1)(0.5)(0.5) = 0.25$. If you flip a fair coin ten times, the number of heads can range from 0 to 10, the expected number is $\mu = 5$ and the variance is $\sigma = 2.5$.
  - **Queueing networks** are examples of system where the variability may be substantially different as a function of parameters, so metamodels of performance as a function of these parameters may exhibit highly heterogeneous variance in some form other than proportional errors. For example, a single-server queue (like a single line for a car wash) can be characterized by its traffic intensity—the arrival rate λ and the service rate μ. If λ is much less than μ (notationally, $\lambda \ll \mu$), the arrival rate is much less than the service rate. Customers will then consistently spend very little, if any, time waiting for service. Think of the car wash when thunderclouds are looming: the carwash may be capable of serving 12 cars per hour, but if they arrive at a rate of only 1 per hour it will be rare to have much of a line. On the other hand, if λ is only slightly less than μ, then the expected customer waiting time is very large. If $\lambda = \mu$, then for certain common distributional assumptions the expected customer waiting time is infinitely large. This does not just mean that regression metamodeling cannot make a reasonable assumption of constant variance as a function of λ/μ—it has serious practical implications as well. One should never seek to have a service system where every server is fully utilized. There needs to be slack in the system to accommodate random variability in arrival or service times.

Causality. Last but not least is the idea of causality, which can be problematic in models of observational data. Despite the mantra that "correlation is not causation," regression terminology makes it natural to act as if changes in X are the cause of changes in Y. However, it could be that changes in Y are what cause changes in X, as when X is the number of umbrellas carried and Y is the rainfall. While the number of observed umbrellas on a given day is probably a good predictor of the amount of rain, umbrellas do not cause precipitation. It could also be that changes in either or both X and Y are (partially) attributable to some lurking variable that affects them both but has been omitted from the analysis. For example, if Y is reading proficiency and X is weight for a group of schoolchildren, age is a lurking variable—it would be a mistake to either force-feed the children to attempt to improve their reading skills, or to provide additional tutoring to attempt to help them grow up faster. Lurking variables also give rise to spurious correlations—such as the fact that the rabbit population in Australia and the London Stock market prices both exhibited exponential growth over an extended period of time. Another possibility is that no underlying relationship exists but we happen by chance to get a statistically significant result because of the built-in error rate of statistical models. This is a concern when data mining—if you evaluate all $\binom{k}{2}$ pairings of k columns from a database where k is large, you will inevitably find spurious correlations.

We hope these examples clarify that the simple regression formulation in equation (4.2) is in truth a *model* of reality. It is based on embedded assumptions that may not hold in the real-world setting where we collect the data.

4.2.1 When we know the model

Sometimes, we might have theoretical knowledge about the underlying system based on well-established principles from science or engineering. For example, the distance d traveled by an object that starts at rest and accelerates at constant rate a for time t in a vacuum is described by the formula:

$$d = \frac{1}{2}\left(at^2\right). \tag{4.5}$$

On Earth, gravity gives an acceleration of $a = 9.81$ meters per second^2, or 32.2 feet per second^2. Equation 4.5 is based on a single quadratic effect t^2 with a coefficient equal to $a/2$, and has no intercept or main effect. If you found yourself suddenly teleported to another planet, along with a vacuum-filled tube and a stopwatch, you could estimate the gravitational constant on the new planet. (Of course, you might want to look for sources of breathable air, water, food, and shelter first.) If your stopwatch isn't completely accurate, or the vacuum isn't perfect, or there are slight discrepancies between when you start to drop the ball and when you start the timer, this might account for the error terms in a regression model. Even knowing that measurement errors probably exist, you may not know the theoretical distribution of those errors. However, based on repeated experiments you could fairly rapidly come up with a good idea of the local gravity.

We can reverse this relationship by using a computer to generate data that come from equation (4.2), or any other analytical model, simply by specifying the appropriate terms, values for the true β coefficients, as well as information about the error distribution. In such cases, looking inside the computer code is like looking into a crystal ball—we regard the theoretically based equations as the ground truth.

4.2.2 When we don't know the model

The cases where we truly know the form of the model tend to be rare. One major difference between the fields of probability and statistics is that in probability we assume ground truth and are then able to make rigorous analytical statements about how samples will behave, while in statistics we are given data and try to make inferences (or gain insights) about the underlying ground truth.

For most simulations, the direct input/output relationship is more complicated than a simple analytic structure such as equation (4.2) or (4.5)—otherwise we wouldn't bother to simulate it. One class of problems arises when it just might be too hard to try to figure out the theoretical distribution. For example, what if our errors aren't any of the standard distributions, but follow something like a shifted exponential distribution? Even worse (from a mathematical standpoint), what if we want to examine errors that arise from the ratio of two non-negative random variables, like a truncated normal distribution divided by a shifted exponential?

A second class of problems involves simulations that mimic some process evolving over time. These include discrete event simulations (DES) and discrete time simulations, also known as time-stepped simulations. Agent-based simulations can be implemented using either paradigm. For these types of computational models, we can look at the code to see how the individual pieces work—but we have to run it to get a sense of the input/output relationships. The response surface is an implicit function that maps inputs to outputs, rather than an explicit one, and that mapping must be estimated from observed data.

Note that historically, there was a tendency to favor analytical formulas over simulation models, even if this meant committing "type III errors" (solving the wrong problem). This made a great deal of sense in the era pre-dating modern computers, where closed form formulas permitted some analysis of problems that would otherwise be completely intractable. However, with modern computers many previously intractable problems are *computationally tractable*—we can simulate them and get answers as precise as we want. As one example, graphs may greatly enhance our understanding of complicated equations. If the confidence intervals for simulation estimates are narrower than the width of the line that's drawn, then it doesn't matter if the solution is analytically tractable or not. This idea means you can avoid using "dead statistician" distributions for the sake of mathematical convenience or necessity. For a more in-depth philosophical discussion of these ideas, see Lucas et al. (2015).

Going back to non-simulation data, it's often the case that we do not know the actual form of the relationship between the inputs and outputs. Still, people often use parametric equations as a model for, or way of, describing these relationships. Regression metamodels can be used to fit the relationships once they have data. The results are relatively easy to interpret and regression techniques are flexible enough to fit a wide variety of response surfaces.

4.3 Multiple Regression Metamodels

In multiple regression we fit the response as a function of more than one factor. The metamodel can vary based on the types of data of both the factors and the responses.

4.3.1 Quantitative Xs and Y

As a refresher on multiple regression, let's start by assuming that Y and the Xs are all quantitative variables. They can either be continuous measurements (e.g., height, weight, reaction time) or discrete (e.g., counts of items).

A few types of models are so frequently used that they have been assigned names.

A *main-effects model* has the following form:

$$Y = \beta_0 + \beta_1 x_1 + \ldots + \beta_k x_k + \varepsilon = \beta_0 + \sum_{i=1}^{k} \beta_i x_i + \varepsilon, \tag{4.6}$$

where the εs are often assumed to be independent and identically distributed (i.i.d.) random errors with a mean of zero.

A full *second-order model* adds both quadratic effects and two-way interactions:

$$Y = \beta_0 + \sum_{i=1}^{k} \beta_i x_i + \sum_{i=1}^{k} \beta_{i,i} x_i^2 + \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \beta_{i,j} x_i x_j + \varepsilon, \tag{4.7}$$

though it's best to fit this equation after centering the quadratic and interaction terms for computational stability and ease of interpretation of the individual factor effects:

$$Y = \beta_0 + \sum_{i=1}^{k} \beta_i x_i + \sum_{i=1}^{k} \beta_{i,i} (x_i - \overline{x}_i)^2 + \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \beta_{i,j} (x_i - \overline{x}_i)(x_j - \overline{x}_j) + \varepsilon. \tag{4.8}$$

Some software (including JMP and R) will do this centering for you automatically. Quadratic effects are interesting because they can allow for increasing or diminishing rates of change, or indicate the presence of a maximum or minimum point in the response For example, Figure 4.1 shows some sketches of various nonlinear models involving a single X. These types of behaviors are sufficiently common that they have simple names associated with

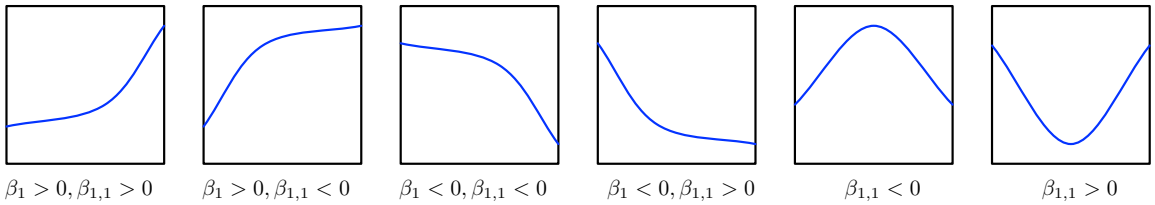


Figure 4.1: Sample relationships that can be fit well by a second-order polynomial. Each sub-plot is a graph of Y vs. X, where the signs of the slope β_1 and the quadratic effect $\beta_{1,1}$ are shown below. Note that we don't need to state anything about the intercept, β_0; we focus on the shape rather than providing units on the axes.

each. From left to right these are: increasing returns, diminishing returns, increasing losses, diminishing losses, unique maximum, and unique minimum. Although the "knees in the curve" or the maximum and minimum appear near the centers of the pictures, that's not a necessity.

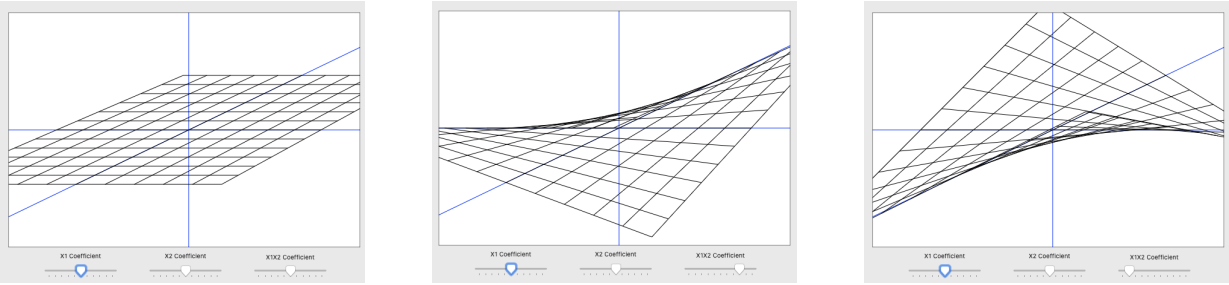
The concept of nonlinearity can be extended very naturally to higher-order polynomials. For example, a full *third-order model* includes main, quadratic, and cubic effects, as well as two-way and three-way interactions.

For any of these metamodel forms, we might not keep all potential terms in our final metamodel. For example, we might have $k = 8$ potential explanatory variables, but perhaps only three of them have an effect that is both statistically significant and practically important. However, if some X is not statistically significant but shows up in a significant interaction or polynomial term, then it's best to keep the main effect in the model as well. The exception is if there's already some theoretical knowledge of the true regression form, such as the gravitational constant estimation described in Section 4.2.1.

Let's discuss interactions in more detail. An interaction means that the whole effect is different from the sum of the partial effects—the magnitude of the impact of a given factor can depend the values of other factors. Interactions can be synergistic. For example, having either a better sensor or a better weapon (but not both) may have very little effect on overall mission effectiveness—even though there's better information provided by the sensor, you might not be able to act on it with existing weapons. However, if having a slightly better weapon combined with a slightly better sensor leads to big improvements in mission effectiveness, that's an example of a synergistic type of interaction. Interactions are not always synergistic. They can be partially redundant. For example, a corporation might consider reducing data spills by adding spam filters or by training personnel to recognize and report phishing attempts. Suppose either intervention in isolation can prevent 80% of the cybersecurity threats, while 85% are caught when both are used.. Only marginal gains occur since so little room for improvement remains, so spam filters and training have largely redundant effects. In other cases, interactions can clash to the level of overwhelming the main effects. Suppose you're trying to design unmanned surface vehicles which can facilitate search and rescue operations in open water conditions. Having greater speed should have a

positive effect, since you can cover more territory. Having higher resolution cameras or other sensors should also have a positive effect, since you are more likely to detect indications of people in the water. In practice, though, the vibrations associated with higher speeds often shake the sensors and negate their higher resolution capabilities. The net effect can actually be fewer detections when you invest in both strategies.

Interactions are also linear, but they make the response surface more complex. Figure 4.2 shows this by mapping a grid across the response surface Y as a function of X_1, X_2, and the X_1X_2 interaction.



(a) No interaction. **(b)** Positive interaction. **(c)** Negative interaction.

Figure 4.2: Response surfaces without and with interactions. The main effects are $\beta_1 = \beta_2 = 0$ in all plots. The grid lines show that a linear change in either x_1 or x_2 leads to a linear change in y.

People looking at Figures 4.2b and 4.2c are often surprised by the statement that this is linear, but an algebraic proof is straightforward. We'll start with Equation 4.7 for two factors, but exclude the quadratic terms:

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{1,2} x_1 x_2 + \varepsilon.$$

If we pick a particular value x_2' for x_2, this becomes

$$\begin{aligned} Y &= \beta_0 + \beta_1 x_1 + \beta_2 x_2' + \beta_{1,2} x_1 x_2' + \varepsilon \\ &= \beta_0 + \beta_1 x_1 + (\beta_2 x_2') + (\beta_{1,2} x_2') x_1 + \varepsilon. \end{aligned}$$

Note that the parenthetically grouped terms are constants, since x_2 is being held constant at x_2'. This allows us to regroup terms as follows, where again, parenthetic terms are constant:

$$\begin{aligned} Y &= (\beta_0 + \beta_2 x_2') + (\beta_1 + \beta_{1,2} x_2') x_1 + \varepsilon \\ &= \beta_0' + \beta_1' x_1 + \varepsilon, \end{aligned}$$

where the end result is clearly linear but has coefficients that have been adjusted based on the value of x_2. The equivalent result showing linearity holds true if x_1 is the factor held constant, and is left as an exercise for the reader.

Recall that metamodeling provides three things:

- A description of the deterministic part of the relationship in the form of a metamodel, i.e., the equation for $\hat{y}_x$.
- A characterization of the uncertainty present. If we are willing to assume that the underlying error variance σ^2 does not depend on the X values, then MSE or RMSE have natural interpretations of estimates of σ^2 or σ, respectively. Alternatively, because heterogeneous variance is pervasive in simulation, we can fit a separate metamodel of σ_x if we conduct a data farming experiment with two or more replications.
- A set of statistics describing the strength of the relationship. The overall statistical significance of the regression metamodel is denoted by a single p-value based on an F-statistic that compares the fitted metamodel to the simplistic null hypothesis metamodel that $\hat{y} = \bar{y}$. Remember that a small p-value means that the metamodel *as a whole* is statistically significant—not that every term in that metamodel is statistically significant. Finally, the *coefficient of determination* R^2 (sometimes called the *multiple correlation coefficient*) is an analog to r^2 that works for describing the practical importance of the relationship for multiple regression. Recall that $0 \leq R^2 \leq 1$, where 0 indicates no linear relationship and 1 indicates a perfect fit. Mathematically,

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y}_i)^2}$$

  and can be interpreted as the proportion of variability in the observed data that is "explained" by the metamodel. The air quotes around explained are necessary when interpreting metamodels fit to observational data, as per our earlier discussion about causality. However, in data farming experiments they are not needed, as changes in the response can be causally attributed to changes in the factor levels across design points (or bugs in the code triggered by such changes). We will discuss some of the diagnostic implications of R^2 in Section 4.6.

4.3.2 Qualitative and quantitative Xs, and quantitative Y

Given modern computers, there is no reason we can't do multiple regression with a mix of qualitative and quantitative Xs, something that was very hard when it had to be done by hand. Here's a quote from a paper written in 1938 about the "relatively new Analysis of Variance" method.

> "The bulk of the calculations consists of adding numbers from a table of squares and can be done by any careful assistant that can operate an adding machine" (Reed, 1938).

Analysis of Variance (ANOVA) is actually a technique used for testing hypotheses about means—not about variances. The name came about because statisticians were trying to find ways of looking at the overall variability in a set of data, and seeing whether or not they could attribute part of it to an independent variable with a fixed number of levels. Regardless of the confusion often caused by the name ANOVA, it has become embedded in the literature and we are stuck with it.

When done manually, this technique required a lot of care and was very prone to error. If you've ever tried adding up (say) 50 numbers one-at-a-time on a mechanical calculator, you will find that it's hard to do without making a mistake. The term "computers" was originally used to describe humans performing these tedious and repetitive calculations—usually women (Garber, 2013)—up through World War II. Fortunately, ANOVA contains a straightforward method for checking the consistency of answers by calculating three types of sums-of-squares, and checking whether the first two add up to the third. This was valuable for detecting and preventing manual errors in calculations, since it flagged work that would need to be redone, but it added to the tedium and time delay before research questions could be addressed.

With modern computing, many people find it easier and more natural to view ANOVA as a special case of multiple regression, where independent variables are qualitative instead of quantitative. Since regression can handle both types of data, we can construct regression metamodels involving a mix of them. Consequently, while many DOE textbooks describe analysis techniques based on ANOVA tables, in this book we stick with regression as a flexible, unifying approach for analysis. The exception is the F-statistic and p-value for the overall regression, which are often presented in an ANOVA table. However, we also present some graphs and discussion that may be more suitable when one or more of our factors is qualitative.

Indicator variables

One of the simplest forms of categorical variables is a binary variable that takes on either the value 1 or the value 0. This is called an *indicator variable* (or *dummy variable*) when the 1 and 0 represent 'true' and 'false' for some condition. For example, suppose we're interested in the monthly rent paid by undergraduate students at Somestate College. If we have a binary variable X_1 called 'upperclass' in our data set, students who are frosh or sophomores are given a value of 0, and students who are juniors or seniors are given a value of 1. We might also have information recorded about the distance to campus, in miles (X_2). With this in mind, we could fit different types of metamodels of rent (Y). The simplest metamodel is just to fit a mean value, i.e., to ignore anything about how close the students are to campus or to graduation. Mathematically, this is:

$$\widehat{rent} \;=\; \hat{\beta}_0. \tag{4.9}$$

Equation (4.9) may have a lot of unexplained variability. There are other types of models we could fit, including:

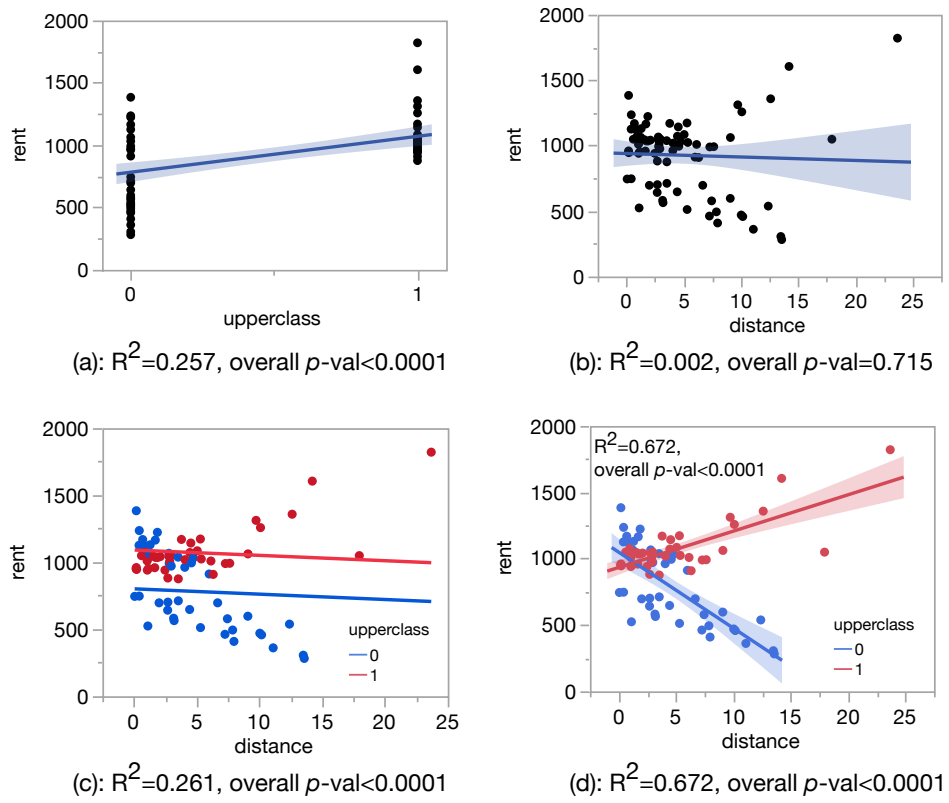
$$\widehat{rent} = \hat{\beta}_0 + \hat{\beta}_1(upperclass), \tag{4.10}$$

$$\widehat{rent} = \hat{\beta}_0 + \hat{\beta}_2(distance), \tag{4.11}$$

$$\widehat{rent} = \hat{\beta}_0 + \hat{\beta}_1(upperclass) + \hat{\beta}_2(distance), \text{ or} \tag{4.12}$$

$$\widehat{rent} = \hat{\beta}_0 + \hat{\beta}_1(upperclass) + \hat{\beta}_2(distance) + \hat{\beta}_{1,2}(upperclass \times distance). \tag{4.13}$$

We fit regressions for each of these using available data. Summary graphs and statistics for the resulting metamodels appear in Figure 4.3. In Figure 4.3(a), corresponding to equation (4.10), $\hat{\beta}_1$ is significantly different from 0.0. This shows that the two groups of students pay different amounts on average. If $\hat{\beta}_1$ had not been statistically significant, we would fail to reject the null hypothesis that rent for underclass students is equal to rent for upperclass



(a): R^2=0.257, overall p-val<0.0001

(b): R^2=0.002, overall p-val=0.715

(c): R^2=0.261, overall p-val<0.0001

(d): R^2=0.672, overall p-val<0.0001

Figure 4.3: Graphical representations of different metamodels of rent paid by Somestate College students, involving a single quantitative variable (distance from campus) and a single indicator variable (upperclass): (a) one indicator variable, (b) one quantitative variable, (c) two main effects, and (d) two main effects and one two-way interaction. Graphs (a), (b), and (d) are from JMP's Graphbuilder, showing the regression lines surrounded by shaded areas representing the 95% confidence intervals for the mean response. Graph (c) comes from JMP's ***Analyze*** → ***Fit Model*** for a main-effects model involving distance and class, and shows only the fitted lines without the confidence bands.

students, and we would end up just making our prediction based on the mean as in equation (4.9). Regardless of what we find (significance or non-significance), we've used multiple regression to conduct an ANOVA.

Figure 4.3(b) shows the fitted metamodel of the form in equation (4.11). This is a simple regression metamodel, although it is not statistically significant (p-value is very high). From this, we would conclude that the rent is not dependent on the distance from campus. Adding the indicator variable, as in Figure 4.3(c), allows the regression metamodel to fit separate *but parallel* lines for each of the two groups of students. In other circumstances, this could lead to a large drop in the estimated error variance so that distance would become statistically significant, although that is not the case for this set of data. However, if we add in an interaction term, both the intercepts and slopes can change for each of the two groups as in Figure 4.3(d). Sometimes these effects are easy to distinguish from just plain scatterplots, leading you to look for a reason that the data might be coming from two or more groups even if you didn't start out with an indicator variable. At other times the groups won't necessarily show up in graphs unless you use color or shape to distinguish them from each other, but they will become evident once you try fitting a more sophisticated regression.

Qualitative factors can have two or more levels that are often easiest to refer to descriptively rather than numerically. Here are some other examples:

- Weekend or weekday, for a data set involving customers at a local restaurant.
- Jungle terrain or urban terrain, for simulated combat operations.
- Queueing priority rules of first-in/first-out (FIFO), last-in/first-out (LIFO), or shortest processing time first (SPT) for a queueing system.
- "Goldilocks" temperature categorizations or whether there was precipitation, for data about event attendance for public activities such as concerts, parades, or voting.

Note that our third example has three categories of queueing disciplines, and Goldilocks liked to categorize temperatures as "too hot", "too cold", or "just right." There is no reason we can't expand the number of categories beyond two levels in regression modeling, and there may be tangible benefits to doing so.

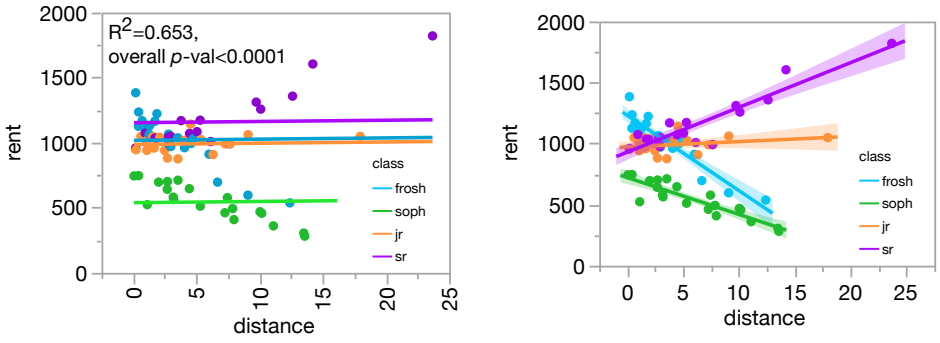
Let's apply this concept to our Somestate College example. Perhaps the underclass/upperclass categorization isn't detailed enough for us. We might be interested in seeing whether there are differences among frosh, sophomores, juniors, and seniors, so we can set up a factor "class" with four levels. This is called a *qualitative* or *categorical* factor; we might also refer to the class levels as *nominal* or *ordinal* data. It might be that not all of these levels are associated with truly different rents, but that's what our statistical software will help us investigate. There are two things to keep in mind.

- The mechanics of fitting multiple levels for a single variable is to create binary indicator variables for all but one of the categories, since that is enough to completely define who's who. For Somestate College, if our three categories are frosh, sophomores,

and juniors, then any student with frosh=0, soph=0, and jr=0 must be a senior. It doesn't matter what category is left out, the model fitting and predictions work equally well, but you must be aware of which was omitted to correctly interpret the results. Interactions involving one or more indicator variables continue to be fit to the cross-product of the respective variables. Interactions between indicator variables and quantitative variables can also be fit. However, note that it does not make sense to try to fit polynomial effects to binary valued variables.

- With modern statistical software packages, the task of setting up the indicator variables all takes place under the hood. You can simply include qualitative variables as a potential explanatory term, along with interactions involving qualitative variables. The software will take care of fitting the regression and reporting the results appropriately. In general, a qualitative factor with ℓ levels will require $\ell - 1$ d.f. for estimating main effects in the metamodel. Similarly, we need an additional $\ell - 1$ d.f. for estimating interactions between a qualitative factor with ℓ levels and a single quantitative factor (regardless of levels). If we have two qualitative factors with ℓ_1 and ℓ_2 levels, respectively, we need an additional $(\ell_1 - 1)(\ell_2 - 1)$ d.f. if we want to fully estimate the two-way interactions for these two factors.

Figure 4.4 shows examples of metamodeling using a qualitative factor with more than two levels, in addition to a quantitative factor. In Figure 4.4(a), we see the results of fitting a main-effects model for rent as a function of one quantitative factor (distance) and one four-level qualitative factor (class). Without interactions the four lines corresponding to the four classes are always parallel. The slopes of the lines are close to zero, but not actually zero.



(a): R^2=0.653, overall p-val<0.0001 (b): R^2=0.939, overall p-val<0.0001

Figure 4.4: Graphical representations of different regression metamodels of rent paid by Somestate College students, involving a single quantitative variable (distance from campus) and a single four-level qualitative variable (class): (a) two main effects, and (b) two main effects and one two-way interaction. Graph (a) comes from JMP's ***Analyze*** → ***Fit Model*** for a main-effects model involving distance and class; it shows only the fitted lines, not the confidence bands. Graph (b) is from JMP's Graphbuilder, showing the regression lines surrounded by a shaded area representing the 95% confidence intervals for the mean responses.

Adding the interaction between distance and class allows a separate line to be drawn for each class. Figure 4.4(b) shows that even though frosh and juniors are very similar in terms of their average rent, they have very different behavior in terms of rent as a function of distance from campus. We have not included a quadratic for distance in either metamodel—if we did, this would allow the lines to be curved rather than straight.

4.4 Partition Tree Metamodels

Regression metamodels can be very informative and helpful for prediction, but two potential downsides are that they may be overly complicated for a non-technical audience, or overly detailed for making model predictions that withstand scrutiny. We can think of regression metamodels that result from data farming experiments as attempts to summarize tens or hundreds of thousands of data points into a few metamodels, each involving a low number of terms. Humans brains can't deal with massive amounts of raw data, so metamodeling facilitates our understanding by summarizing that data in a comprehensible way.

On the other hand, graphics are generally accessible to non-technical viewers, so they can be very informative and useful for generating insights. However, they may not make it easy to quantify those insights.

Suppose we want something in between these two. Partition trees are a nonparametric approach that describes relationships between factors (Xs) and the response (Y) by identifying similar groupings within a set of data. In a data farming context, this offers a possible bridge between quantitative rigor and visual insight. They can be used both in lieu of or as part of multiple regression metamodels.

The concept is simple, but so computationally demanding that it was ignored prior to the availability of modern computing. The data starts in one large group, and the mean and standard deviation of Y are computed. Then, for each of the possible Xs, the software evaluates the value of every potential binary split in terms of its increase to R^2, and selects the one that creates the largest increase.

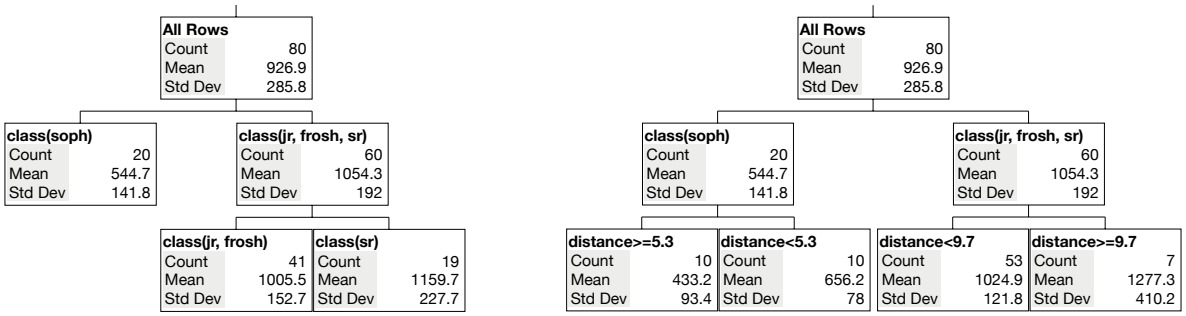
- For quantitative (continuous or integer-valued) factors, this split can occur any place between the minimum and maximum value in the dataset. For example, if the factor `speed` varies from 0.2 to 0.8 in our dataset from a capture-the-flag dataset, the split might occur so one group has data corresponding to `speed` < 0.6 and the other has `speed` ≥ 0.6. One caveat is that each split needs to maintain a minimum size in terms of the number of observations within each category created by that split. Statistical software packages have built-in defaults that can be overridden by the user. In Section 4.8. we will discuss this in more detail in the data farming context.
- For qualitative ("nominal") factors, this split can occur by lumping any of the different factor levels together. For example, if we have sensor types A, B, C, D, and E, it might

be that the partition tree lumps A and D together in one group {A, D}, and B, C, and E in a second group {B, C, E}.

The net result is that for each of the possible Xs there is a "best" place to split to create two groups. These candidate splits are compared, and the one that leads to the most improvement in the fitted model is selected. This divides the data from one group into two smaller groups. Ideally, there will be big differences in the means between the two smaller groups, but the standard deviation within each group will be smaller.

This splitting process can be continued, with the software automatically checking to determine the best split for each of the "leaves" or smaller groups in the regression tree. Just as R^2 never decreases when we add additional explanatory terms to a regression metamodel, R^2 will never decrease when we make additional splits in a partition tree. As before, the goal is to see whether a parsimonious (simple) model can be constructed with a statistically significant and practically useful R^2 value.

Returning to the Somestate University data, Figure 4.5 shows two partition trees. In Figure 4.5a the splits involve class. The results are similar to those seen in Figure 4.4, showing that sophomores pay the lowest average rent by far, while seniors pay slightly more than frosh and juniors. A partition tree on rent involving distance has a harder time fitting the data: $R^2 = 0.102$ after three splits and rises steadily to $R^2 = 0.247$ after ten splits. Both class and distance are potential explanatory variables in Figure 4.5b.



(a) Partition on class, $R^2 = 0.652$ after 2 splits. **(b)** Partition on class and distance, $R^2 = 0.703$ after 3 splits.

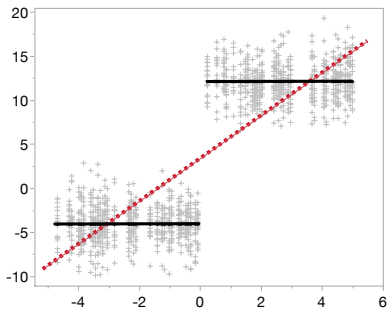
Figure 4.5: Partial partition trees for Somestate University.

As we stated earlier, this technique is nonparametric, i.e., we do not need to *a priori* specify the structural form of the relationships. To contrast this with regression metamodeling, with partition trees we do not explicitly specify terms such as potential quadratic or interaction effects. Each of the factors X_1, X_2, ... can show up in multiple splits throughout the tree. If there is a factor X_i that affects the response Y in a nonlinear way, such as exhibiting diminishing returns, we will see that reflected in the way the means vary from leaf to leaf. If there are strong interaction effects, then splits on one branch may be quite a bit different than splits on another branch—in terms of either the factors used to make the splits, the values at which the splits occur, or the differences in means associated with these splits. This is evident in Figure 4.5b, where the strong interaction is apparent as sophomores pay

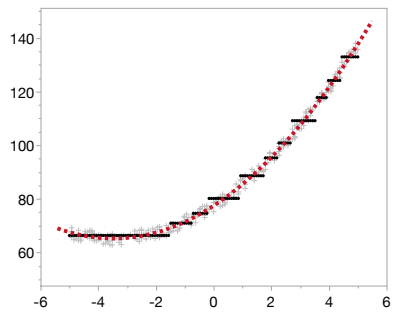
more to live close to campus, while the other classes pay more when living farther away. The right-most leaf in Figure 4.5b has too few points to be further split, so the wide difference in rents among the non-sophomore classes shows up in the leaf's high standard deviation.

Partition trees offer both advantages and disadvantages over multiple regression metamodels, depending on the data.

- Trees are easy for the analyst, who needs only to specify the Y and Xs rather than forcing them to specify a particular functional relationship as, for example, deciding to use a form other than second-order polynomial metamodels in conjunction with stepwise regression. The partition tree approach itself provides the structure.
- Trees are useful for screening a large set of factors to determine which are dominant. Dominance might appear as a single split with a large contribution to R^2, or as the cumulative contribution of many splits throughout the tree.
- Trees never make predictions that fall outside the boundaries of the data, which can sometimes occur with multiple regression. For example, if we fit a polynomial metamodel where the response Y is bounded, such as when Y is the proportion of time we win the capture-the-flag game, a regression metamodel may predict results below 0 or above 1 when speed and stealth are near their boundaries even if it works well for intermediate speed and stealth values.
- Trees are easy to interpret with minimal training or explanation, and work well when presenting results to either technical or non-technical audiences. If decision makers select leaves of the partition tree where desirable outcomes are observed, the paths from the root of the tree to those leaves represent factor thresholds which are associated with the desired end states.
- Trees work extremely well at fitting plateaus in the response, a task which regression handles very poorly. Conversely, partition trees may require an overly large number of splits to fit a smooth function where regression works relatively well. Figure 4.6 illustrates these situations for notional datasets. In Figure 4.6a, the step function in



(a) Partition tree metamodel is better.



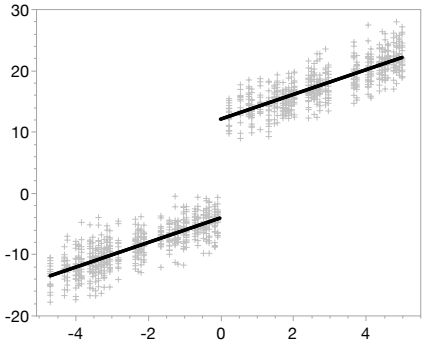
(b) Regression metamodel is better.

Figure 4.6: Comparison of polynomial metamodels and partition trees for two graphs of Y vs. X. Data points appear in grey, the polynomial metamodels are dotted red curves, and the partition tree predictions are black horizontal line segments.

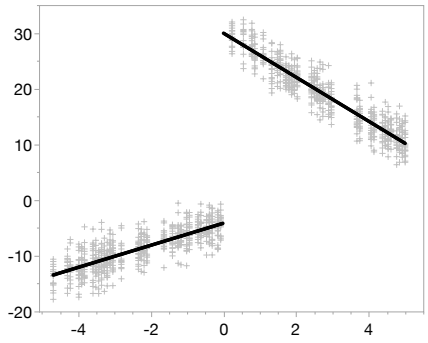
the average response Y is easily fit by a single split in the partition tree, while linear regression does poorly—it takes a very high degree polynomial to provide a reasonable approximation to a step function. Figure 4.6b illustrates the opposite case, where a simple quadratic function is a more parsimonious way to describe the relationship than the partition tree with 10 splits.

4.5 Hybrid Metamodels

Sometimes response surface modeling might benefit from a hybrid metamodel that allows you to capture both step function behavior and smooth relationships in a single framework. For example, in Figure 4.6a it is clear that the response behaves very differently when X is negative than when X is positive. Now suppose that the data on either side of the step exhibit similar slopes, as in Figure 4.7a. In this case, the horizontal line segments from a partition tree would work poorly. However, adding an indicator variable $I_{X>0}$ in addition to X would allow you to capture both the step behavior and the overall linear relationship quite parsimoniously. If the pattern of the relationship also differs on either side of the step, as in as in Figure 4.7b, then include an an interaction between $I_{X>0}$ and X to allow the slopes to differ. The derived factor need not be based on a single factor if, say, you wanted to include the first two splits of a partition tree.



(a) Hybrid metamodel with an indicator to capture the step increase.



(b) Incorporating an interaction between X and the indicator allows the slopes to differ.

Figure 4.7: Hybrid metamodels that incorporate an indicator variable based on the first split of the partition tree. Data points appear in grey, and the metamodel predictions in black.

Figures 4.6 and 4.7 are graphs for a single quantitative X, and are intended to visually illustrate basic concepts. However, those same principles apply for higher-dimensional data, making hybrid models a powerful tool for analyzing complex systems.

4.6 Other Multiple Regression Metamodels

Our goal in datafarming is to use the data obtained from designed experiments to construct metamodels that are useful approximations to the true response surface of the system we're studying. Throughout most of this chapter, we have focused on the use of second-order polynomial metamodels as useful ways to capture relationships between quantitative factors and quantitative responses, because they can capture interaction and nonlinear effects. Not all nonlinear effects are quadratic, but multiple regression can be used to fit other types of nonlinearities as well. We provide examples of different types of nonlinearity, and assert that space-filling designs are an absolute necessity for analysts attempting to detect this behavior and incorporate it into suitable metamodels.

If sampling is done at a small number of levels per factor, there might be large deviations between the true mean response and our metamodel in the regions where we have no samples. Unless you have good reasons—based on known theory or lots of prior evidence—to restrict your model to linear effects, we strongly urge you to consider using space-filling designs. By their very structure they reduce the gaps which are intrinsic to classical designs such as factorials or CCDs with 2 or 3 levels per factor, and they facilitate detection of non-polynomial nonlinearity, such as exponential, logarithmic, logistic, or plateau behaviors.

To illustrate our point, we present a small example comprised of four different functions, all based on a single factor x. The functions are all deterministic to avoid clouding the results with noise, so a single observation at each design point is sufficient. The four response surfaces are:

$$\left.\begin{aligned} f(x) &= 3x^2 - 15x \\ g(x) &= x^3 + 0.3x^2 - 39.75x + 67 \\ h(x) &= \exp(-x) \\ m(x) &= \begin{cases} 150 & \text{if } x < -4.2 \\ 120 & \text{if } x < \frac{15}{7} \\ 0 & \text{otherwise} \end{cases} \end{aligned}\right\} \quad \text{for } -5 \le x \le 5.$$

When we plot these functions, we will play "connect-the-dots" and leave it to the reader to infer any underlying curvature.

We evaluate all four systems by running them at designs with ℓ, the number of levels, equal to 2 through 5, 10, 20, and 50. Each design spans the full range $[-5, 5]$, and ℓ divides the range into equal subranges, i.e., the dps are equally spaced. We illustrate this for the first four designs in Table 4.1, and trust you will quickly see the construction. The results for all four response surface samples are jointly plotted for each value of ℓ.

As a quick reminder, recall that the highest order polynomial curve we can fit based on ℓ distinct dps has order $\ell - 1$. Given two dps we can only fit linear models, with three dps we can fit second order models, and so on. This, combined with our knowledge of ground truth, leads us directly to the weakness of 2-level designs. The results for $\ell = 2$ can be seen

Table 4.1: Equally spaced design point sets spanning the range $[-5, 5]$ for $\ell \in \{2, \ldots, 5\}$.

ℓ	design points
2	{-5, 5}
3	{-5, 0, 5}
4	{-5, -1.667, 1.667, 5}
5	{-5, -2.5, 0, 2.5, 5}

in Figure 4.8. The best we can do is fit all four functions as a straight line, and lo and behold, all four lines are virtually indistinguishable. In this instance none of the response surfaces are actually linear, but if we didn't know the ground truth and this was the result observed by studying a parsimonious 2-level design, we would conclude that there is no visible difference between the four systems' performances.

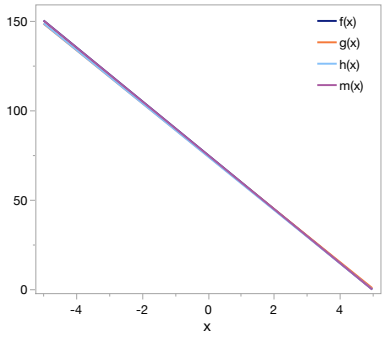
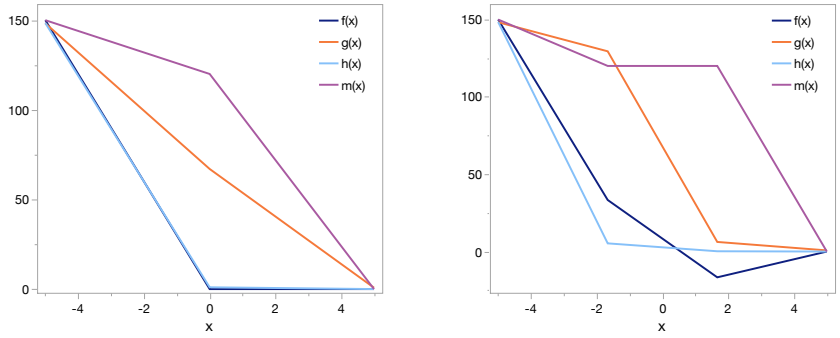


Figure 4.8: Sampling at 2 levels.

In fitting a line with two observations, we don't have any degrees of freedom so most statistics packages will decline to report R^2. However, note conceptually that there is no unexplained variance—each of our four lines goes through its two observations with perfect accuracy. Does this reflect that we have four metamodels that are congruent with the four response surfaces? No, it quite literally just means that we can always draw a straight line between two points. Could we go ahead and fit a quadratic or cubic to the two design points? There is no mathematical basis for doing so, and even if somebody insists that the underlying relationship is known to be quadratic there are an infinite number of quadratics that could go through any two given points.

This is where the analysis flexibility associated with the space-filling designs of Chapter 3 comes into play. We see what happens as we bump ℓ to 3 or 4 in Figure 4.9. An analyst looking at Figure 4.9a would still be unable to distinguish between $f(x)$ and $h(x)$, but $g(x)$ and $m(x)$ are clearly distinct from all others. However, with 3 levels in the design we're limited to second order fits, and based on our insider knowledge we know that's not going to work for three of the four response surfaces. Note that aside from a slight kink in the middle, which would be easily masked by randomness if these were stochastic systems, $g(x)$ still looks like it could be linear.

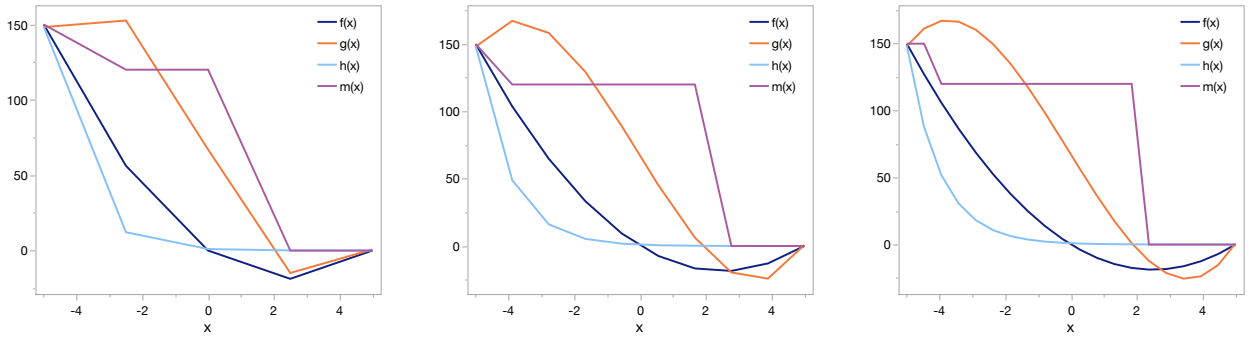


(a) 3 levels **(b)** 4 levels

Figure 4.9: Effects of adding 3 and 4 levels.

As we progress to 4 levels in Figure 4.9b, we finally have all four response surfaces showing distinct behavior. An experienced analyst might look at how $f(x)$ diminishes but then starts increasing again, and tentatively identify that as quadratic. Similarly, noting that $h(x)$ is monotonically decreasing but the magnitude of the decrease diminishes from left to right could lead to tentative identification as a negative exponential. In both cases the tentative choices would be right, but it's not a sure thing. Both $g(x)$ and $m(x)$ appear to have two turnovers, a behavior exhibited by cubics. But we still only have 4 dps, and thus are limited to third order polynomial levels of complexity.

The picture becomes progressively clearer as we increase ℓ through values of 5, 10, 20, and 50 in Figure 4.10. The odd-one out remains $m(x)$, but it is obvious by Figure 4.10b that a partition tree would be a reasonable analysis technique to apply for that system. The main improvement as we work our way up to $\ell = 50$ is the resolution to which we can isolate and identify the breakpoints in $m(x)$.

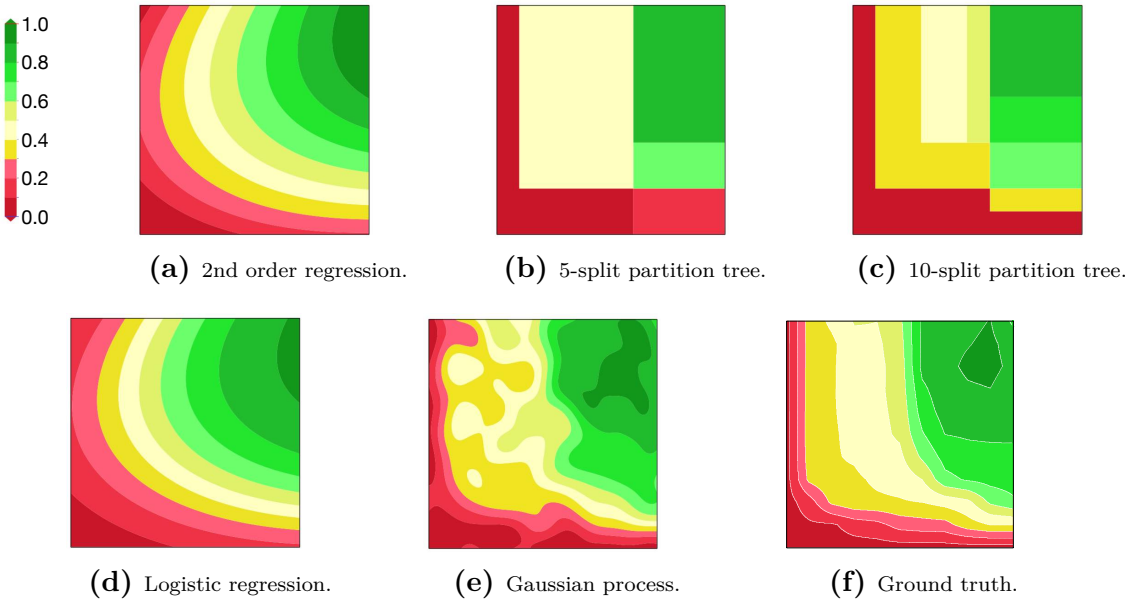


(a) 5 levels **(b)** 10 levels **(c)** 20 levels

Figure 4.10: Adding more levels gives better resolution but diminishing returns to scale.

4.7 Other Types of Metamodels

The coverage in this chapter is not meant to be comprehensive, but instead to give a practical introduction to simple approaches that have been shown to work well in the data farming context. There are a few other metamodeling techniques that may be useful. For illustration purposes, we first provide contour plots for several metamodels of an experiment involving the capture-the-flag simulation, and then touch on some of the key characteristics of these metamodels. All of the plots are based on 100 replications of an 11^2 full factorial design, for a total of 12,100 simulated games. In Figure 4.11, the fitted metamodels were evaluated on a fine grid to reduce the interpolation required when plotting the contours. The outcomes for the home team are represented using a color scheme ranging from bad (red) to good (green). Figure 4.11a shows the results of fitting a second-order polynomial regression model to the data, while Figures 4.11b and 4.11c show those for a partition tree with 5 and 10 splits, respectively. These types of metamodels were discussed earlier in this chapter. The other plots show contours from a *logistic regression* metamodel (Figure 4.11d) and a *Gaussian process* metamodel (Figure 4.11e), respectively, both described below. For comparison purposes, the "ground truth" appears in Figure 4.11f.



(a) 2nd order regression. **(b)** 5-split partition tree. **(c)** 10-split partition tree.

(d) Logistic regression. **(e)** Gaussian process. **(f)** Ground truth.

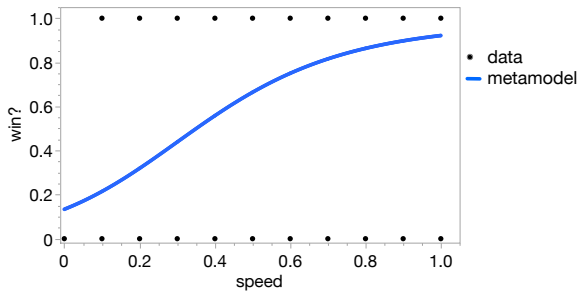
Figure 4.11: Contour plots of metamodels for predicting the P(win) for capture-the-flag, based on speed (x-axis) and stealth (y-axis), from 100 replications of a 11^2 full factorial. A plot for the ground truth (from the simulation) is provided for comparison purposes. Speed and stealth both range on standardized scales from 0 (low) to 1 (high). All plots use the same color scheme, ranging from dark red (lowest P(win) in the lower left corner) to dark green (highest P(win) near the upper right corner).

Looking over the various contour plots, we see they all provide similar qualitative information: neither speed nor stealth suffices on its own, we do not need to push both speed and stealth to their highest values to get a good outcome, and there are many combinations of speed and stealth that yield intermediate outcomes.

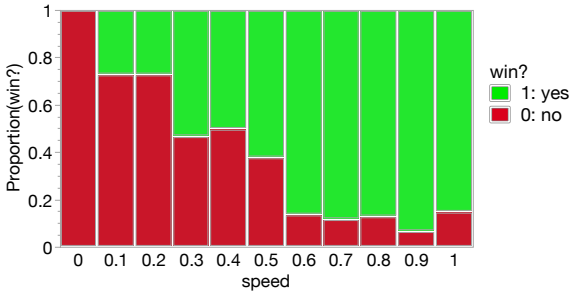
Logistic and multinomial logistic regression

Logistic regression metamodels can be useful for binary-valued responses, such as the winner or loser of a capture-the-flag game. For a single factor X, a logistic model estimates the probability of a success using a curve that is bounded between 0 and 1. The location and steepness of the change in the curve indicate the strength of the factor's effect. If only a linear term is used, the resulting metamodel is S-shaped and monotonic.

More generally, a logistic regression metamodel is a bounded surface that can be fit for multiple factors. Let's revisit the capture-the-flag analysis as a concrete example of why this approach may be desirable when dealing with binary outcomes. Figure 4.12a shows that graphs of the raw data are not particularly informative with multiple replications. Since there are at most two outcomes at each dp, outcomes are overlaid on each other and we get no visual representation of relative frequency. The proportions of the outcomes at each dp, as shown in Figure 4.12b, form a basis for constructing the metamodel. Despite some evidence of lack-of-fit, the resulting metamodel and all five individual terms (two main effects, two quadratics, and one interaction) are statistically significant, and the $R^2(U)$ (pseudo-R^2 used for logistic regression) is 0.246.



(a) Plot of logistic regression, where $y = 1$ for a win and $y = 0$ for a loss. The metamodel surface shows the fitted probability of a win as a function of speed and stealth.

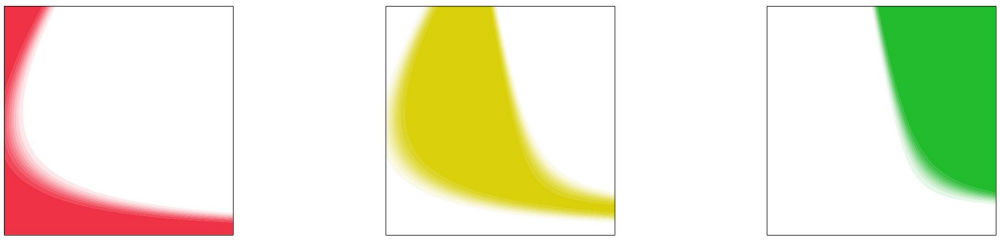


(b) Proportions of wins associated with the simulation data, used to fit the logistic model in Figure 4.12a.

Figure 4.12: Examples of a logistic metamodel for capture-the-flag as a function of speed. For illustrative purposes, we fix stealth=0.9 for this plot.

Logistic and multinomial logistic metamodels can be advantageous to polynomial regression metamodels. Specifically, predictions made using these models are guaranteed to fall within constrained boundaries. For example, the predicted probability of winning at capture-the-flag ($\hat{P}(win)$) always satisfies $0 \leq \hat{P}(win) \leq 1$ for logistic regression, while polynomial regression can yield predictions below 0 or above 1. This actually occurred in the metamodel of Figure 4.11a. Some of the predictions for that metamodel exceed the 0/1 bounds, even though the contour plot appears to be very similar to that of 4.11d.

Similarly, multinomial logistic models can be used to fit qualitative responses with three or more categories. The results of a logistic regression for predicting a poor, intermediate, or good outcome appear in Figure 4.13.



(a) Pr(red): poor team setup. **(b)** Pr(yellow): intermediate team setup. **(c)** Pr(green): good team setup.

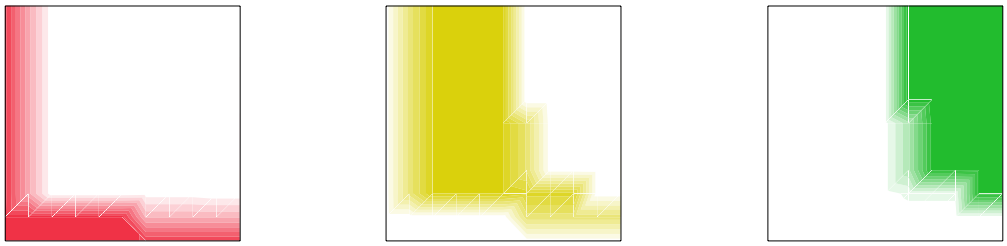
Figure 4.13: Contour plots of multinomial logistic metamodel for fitting a qualitative response with three levels (red=poor, yellow=medium, and green=good) for capture-the-flag. Each plot indicates the predicted likelihood of observing the respective category as a function of speed (x-axis) and stealth (y-axis). The preponderance of shaded areas correspond to Pr(occur)=1.0.

Some further examples where multinomial logistic regression may be suitable follow.

- A metamodel of a COVID test corresponding to a true positive, true negative, false positive, or false negative result.
- The status of a network as either up, down for scheduled maintenance, or down because of an unscheduled event. Unscheduled outages could be further categorized by type, such as mechanical failure, denial-of-service attack, or adverse weather conditions.

We remark that partition trees remain a viable and easy-to-understand metamodeling technique for qualitative Ys. For example, Figure 4.14 shows the contours for a partition tree for the red/yellow/green (poor/intermediate/good) color coding. This is qualitatively similar to those for the multinomial logistic metamodel in Figure 4.13 and, like the logistics metamodels, all predictions for the Pr(occurrence) of a particular category are constrained to be between 0 and 1.

The effect of stealth from the partition tree is easier to explain than that for the logistic metamodel, since Figure 4.14a shows that increasing stealth never hurts the team. In contrast, Figure 4.13a indicates that at low speed increasing stealth initially helps, but then hurts the team performance—an artifact of the particular parametric form used for that model.



(a) Pr(red): poor team setup. **(b)** Pr(yellow): intermediate team setup. **(c)** Pr(green): good team setup.

Figure 4.14: Contour plots from a 10-split partition tree metamodel for a qualitative response with three levels (red=poor, yellow=intermediate, and green=good) for capture-the-flag. Plots (a)–(c) indicate the predicted likelihood of observing the respective category. The preponderance of shaded areas correspond to Pr(occur)=1.0.

Contour plots are not necessary for the partition tree—the tree itself is easy to interpret, as Figure 4.15 shows. We can see the success of the categorization by color coding the data and noticing that many of the leaves are dominated by a single color.

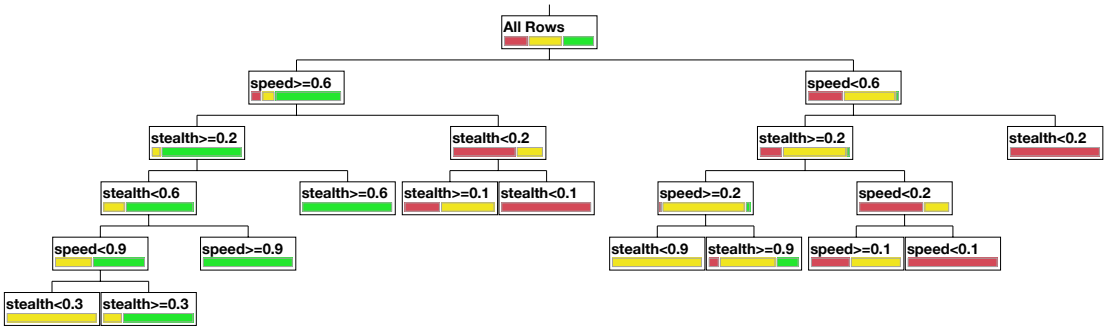
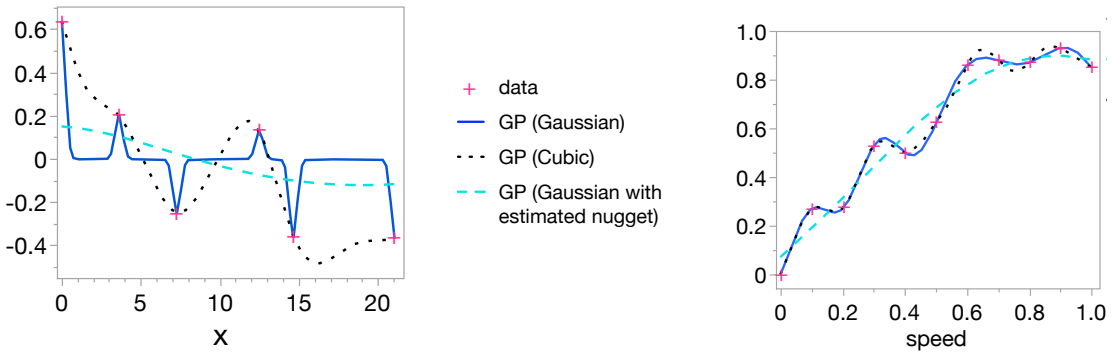


Figure 4.15: 10-split partition tree metamodel for a qualitative response with three levels (red=poor, yellow=intermediate, and green=good) for capture-the-flag. The metamodel yields $R^2 = 0.801$.

Gaussian process metamodeling

Gaussian process (GP) metamodels are another class of metamodels that are popular within some communities, particularly in scientific simulations and some machine learning applications (Rasmussen and Williams, 2006). Originally developed for spacial applications, a GP metamodel fits a spatial correlation metamodel to the data, where the correlation between any two responses (y_i and y_j) diminishes as the distance between the corresponding x_i and x_j increases. This correlation function is often called the *kernel*. One advantage of GP metamodels is their flexibility: unlike polynomial metamodels or other types of generalized linear models, the analyst need not postulate a particular functional relationship between the simulation inputs and the (average) output. Another potential advantage is their provision of a metamodel-based estimate of the prediction errors even if the simulation itself is deterministic.

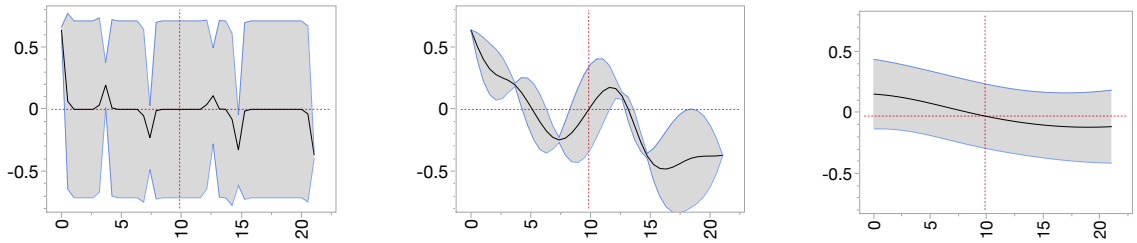
However, GP metamodeling has its own set of challenges. As Figure 4.16 shows, you can get radically different metamodels by changing the kernel or other parameters of the GP metamodel fitting algorithms. Figure 4.16a provides metamodel fits to a small dataset (n=6) under three conditions: using a Gaussian correlation function, a cubic correlation function, or a Gaussian correlation with an estimated *nugget* parameter to avoid singularity in matrix conversion. Here we see that the GP (Gaussian) metamodel quickly reverts to the overall mean over much of the interval, while the GP (cubic) is much smoother; both these metamodels hit the actual data values with no error. The GP (Gaussian with estimated nugget) is very smooth but does not go through the actual datapoints. Figure 4.16b shows these three versions of GP metamodels fit to the capture-the-flag data when speed is set to 0.90, as in Figure 4.12. Here, the GP (Gaussian) and GP (cubic) are quite similar and both hit the actual data points. Once again, estimating the nugget yields a much smoother metamodel.



(a) Artificial dataset. **(b)** Capture-the-flag experiment.

Figure 4.16: Gaussian process metamodels using three different correlation functions on two different sets of data: (a) a small artificial dataset ($n = 6$), and (b) 100 replications at 11 dps data from the capture-the-flag experiment results when stealth is fixed at 0.90. Note that the data graphed in (b) represents the proportion of wins for the corresponding speed.

The uncertainties associated with these metamodel parameterization choices also differ, as shown for the small artificial dataset in Figure 4.17. Those unfamiliar with GP metamodeling may not even be aware that these differences exist, as widely-available commercial and open-source software package use different default algorithms under the hood (Erickson et al., 2018) and many users will just accept that those defaults must represent good choices.



(a) Gaussian. **(b)** Cubic. **(c)** Gaussian with nugget.

Figure 4.17: Gray areas show the metamodel-based prediction error estimates for the GP metamodels of Figure 4.16a fit to a small artificial dataset ($n = 6$) for three correlation functions.

GP metamodeling was initially developed for deterministic simulations when data are very limited, and the self-reported prediction errors can be used to suggest where next to sample to improve the metamodel's accuracy. While they can also be used when the simulation is stochastic or datasets are large, GP metamodel fitting under such conditions involves extensions of the method that may require additional algorithmic parameters or data segmentation into more manageable chunks.

Computationally tractable ways of fitting and especially interpreting GP metamodels involving qualitative factors, particularly when the number of factors or dps is large, is an active area of research. For more on the use of GP models for stochastic simulations, see, e.g, Ankenman et al. (2008, 2010), Binois et al. (2016), or Chapter 5 of Kleijnen (2015).

4.8 Data Farming Considerations

There are a few additional steps you might consider when you are dealing with data from a designed experiment, rather than observational data. Chapter 5 will describe in more detail different types of analyses we suggest when exploring the results. In this section, we focus on some general approaches for coming up with decent metamodels. Note that we are not delving into detailed metamodel diagnostics, though in some instances we indicate why certain classical statistics may be less relevant when data farming than when simply data mining. We begin with a brief discussion of our overall philosophy for metamodeling, and then move on to provide some specific tactics.

4.8.1 Metamodeling goals while farming for insights

Recall the "Farming for Insights" PDSA cycle of Figure 1.1 in Chapter 1. The experiment is set up in the PLAN phase, and executed in the DO phase. The results are explored via analysis and visualization tools in the STUDY phase, where metamodeling plays a key role. In the ACT phase, after logging key analysis findings and supporting data, you will either: (i) move to the "Farming for Modeling" cycle if potential bugs are uncovered or questions arise that require additional output from or enhancements to the conceptual model; (ii) stay in the "Farming for Insights" cycle if the simulation model appears to be working properly but you need further experimentation; and eventually (iii) move to the "Insights to Impact" cycle if the results are actionable. The ultimate purpose of metamodeling is to support the development of actionable insights.

Our philosophy, expressed in Sanchez and Lucas (2002) and Kleijnen et al. (2005), is that there are three goals for data farming studies: (i) to develop a basic understanding of a particular simulation model or system; (ii) to find robust decisions or policies; and (iii) to compare the merits of various decisions or policies. These are substantially different from the goals that Santner et al. (2003) proposed for deterministic computer experiments: (i) calibrating the computer model to real-world data; and (ii) making predictions at untried combinations of the input factors, or (iii) optimizing a response. While our metamodels will allow us to make predictions at previously untried dps, our insights may be more qualitative than quantitative in nature. We may be more interested in identifying which of the k factors are impactful, and the nature of that impact (including the presence of interactions and nonlinearities), than in finding a metamodel that yields the smallest possible prediction error for a fixed amount of sampling. For a more specific example, all the contour plots for capture-the-flag in Section 4.7 show that in order to achieve good outcomes, we need at least a small amount of stealth together with a moderately high level of speed. This insight is actionable for deciding how to coach the team. A deep dive into trying to find a metamodel that most closely matches the ground truth might not be worth the effort, particularly since the ground truth of this simulation's behavior is a low-fidelity model of a real world game.

More generally, there is a limit on how much predictive ability is needed when we seek to make effective decisions about a real-world situation. Recall the pizza hut example in Chapter 2. If we accept that our simulation model is an approximation of (future) reality, then we may be more interested in qualitative insights than in quantitative predictions. This is another benefit of simple models. Even simple models may easily end up with dozens or hundreds of factors. The analyst may find that time spent to better understand the underlying response surface for a simple model may be a far better investment than time spent adding detail and complexity to the model. A similar philosophy holds for metamodeling—the analyst may make better progress toward actionable insights using relatively simple metamodels than by using computationally intensive methods to develop and refine complicated metamodels. In practice, it's often the case that specific numerical predictions are less useful than knowing whether to increase or decrease a factor to improve performance, and whether that adjustment might yield increasing or diminishing returns. Similarly, knowing that a knee-in-the-curve exists may be more useful than having a specific prediction of its placement. Both of these capture the overall philosophy that understanding the shape of the response surface may produce more insight, and consequently better analysis, than specific point predictions. This aligns with the observation by Tukey (1962):

> Far better an approximate answer to the *right* question, which is often vague, than the *exact* answer to the wrong question, which can always be made precise.

This highlights a fundamental tradeoff between complexity and insight that is important whether you are the modeler, the analyst, or the decision maker.

The goal is to model the actual response surface, not just the data we have access to. This brings us back to some of the concepts in Chapter 3—particularly the importance of using a space-filling design. Metamodel diagnostics are based on how close the metamodel comes to the observed data, but if there are large gaps in the data, we have no idea how our metamodel is performing in those regions. Space-filling designs address this issue by intentionally filling gaps that sparse designs contain.

4.8.2 Raw, summary, and transformed data

Many of the data sets you used in your basic statistics class may have been small (at most a few hundred observations). Plots get more crowded when you attempt to look at thousands of data points on a single plot, as may happen with large-scale simulation studies or other types of big data. Consequently, you may wish to work with summarizations of the data instead of the raw data.

It's often particularly useful to summarize the results for your responses by design point, i.e., across the replications at that dp. If you conduct n_r replications of a design with k factors, n_d dps, and n_y raw responses, then your initial analysis dataset will have $n_d \times n_r$ rows and at least $k + n_y$ columns: k for the factors, and n_y for the responses. (There may also be extra columns to identify the dp, the random number seed, the date, or other data

related to particular simulation runs.) When summarizing these data by dp, you should at least include a measure of the central tendency (such as the mean or median) and a measure of the spread (such as the standard deviation or interquartile range). If, for example, you calculate the mean and standard deviation at each dp, then your summary dataset will then have n_d rows and $k + 2n_y$ columns. If you include the same terms in, say, multiple regression metamodels for Y and for $\bar{Y}$, these both result in identical βs for those terms. The $\bar{Y}$ metamodel will have a higher R^2—the MSE will decrease by roughly $1/n_r$ and hence the RMSE by roughly $1/\sqrt{n_r}$—but this is not cheating because the two metamodels are fitting different things. Just remember that if you fit the metamodel for the summary data but wish to make prediction intervals for individual runs, you will need to convert the MSE back to the original units. Also, and perhaps more importantly, you should consider the shape of the residual distribution when making such predictions. However, as discussed earlier, for many situations you may be more interested in identifying the key drivers of the means and variabilities of the responses, rather than making detailed predictions. Consequently, the primary practical goal will be to identify the influential factors and terms—and this can be done well with the summarized data.

Slight differences can occur when using summarized data for other types of metamodels. For example, a partition tree excludes splits that would violate the minimum leaf size specification. This can lead to differences in the trees generated from summarized datasets versus raw datasets, since the former aggregate many points into a single observation. If the minimum leaf size is 5, then each leaf in a tree for the summarized dataset must contain 5 or more dps. However, a leaf could be comprised of a single dp in the raw dataset as long as $n_r \geq 5$. A second example is that summarizing binary-valued responses over the replications opens up the possibility of using models intended for continuous-valued responses. Finally, summarization is often a *de facto* requirement for GP metamodeling since GP algorithms become computationally intractable for large datasets.

If you have transformed any factors to construct inputs for your simulation model, make sure you conduct your analysis using the factors. For example, if your queueing simulation takes the arrival rate λ and service mean time μ as inputs, but you created your experiment by specifying λ and the traffic intensity $\rho \equiv \lambda/\mu$ as factors in order to guarantee queue stability, then you should add a column ρ to the dataset and perform your analyses on λ and ρ rather than on λ and μ. This is particularly important if the transformation you performed is nonlinear—as in this queueing example—and will thus negatively impact the carefully constructed orthogonality of the design.

At other times, you might want to combine two or more raw responses to create a new one. For example, instead of reporting the number of customers whose wait time exceeded a threshold t, convert that to a proportion of the customers served with excessive wait times. There are a few other transformations that may be appropriate and are commonly used. For example, some results are more easily understood on a logarithmic scale. Alternatively, if Y involves some type of probabilistic search or detection such as the amount of time required to locate survivors in a disaster relief effort, then $1/Y$ may be a suitable response to analyze.

We assert that some other types of transformations are not appropriate. For example, you may have learned that transforming Y is a good idea if you notice that the residuals are non-normal with non-homogeneous variance. However, we are not in favor of any transformation that cannot be easily interpreted. For example, we prefer a logistic regression metamodel for capture-the-flag win/loss outcomes versus the use of an arcsin transformation on the proportions of wins—the latter is not transparent to decision makers (see, e.g., Wharton and Hui (2011)). Fortunately such transformations are not required. First, for either multiple regression or partition tree metamodels, the effect estimators ($\hat{\beta}$s) are unbiased point estimators of the underlying true βs regardless of the error variance. Second, we often are explicitly interested in assessing how the variability changes as one of our response measures, and consequently should construct a metamodel to study that directly.

4.8.3 Statistical significance and practical importance

Many of the assumptions and analysis guidelines that you learned in your regression class don't directly apply when we move to simulated data. An important difference when fitting metamodels is that statistical significance is not the same as practical importance. With simulation, it is often quick and easy to generate oodles and oodles of data unless your simulation takes a long time to run. Consequently, if we run our simulation model at two or more distinct dps with (some) different factor levels—and we run it for a sufficiently large number of replications—we will find statistically significant coefficients for those factors with probabilities of essentially 1.0. Remember, statistical significance measures our ability to state that an effect is non-zero, and large sample sizes allow us to say that with tremendous confidence even when the effect is minuscule. Even if terms are statistically significant, they may explain such a small fraction of the resulting variability in the metamodel that they are not important—they just add some decimal dust to our predictions. As a result, we often end up leaving out a (potentially large) number of statistically significant terms if they don't clear the bar of practical importance. There is no hard-and-fast rule for making this decision, and there may be more than one "good" metamodel that can be constructed from a set of simulation data, but in general we must consider how to balance metamodel parsimony and metamodel explanatory power when we decide how many terms to keep in the metamodel.

We'll give one short example now. Which of the following metamodels would you prefer to brief to your supervisor or research sponsor?

- Alpha, a metamodel with 3 main effects, 2 quadratic terms, 1 two-way interaction, and $R^2 = 0.85$;
- Bravo, a metamodel with 5 main effects, 4 quadratic terms, 3 two-way interactions, and $R^2 = 0.86$; or
- Charlie, a metamodel with 303 main effects, 202 quadratic terms, 101 two-way interactions, and $R^2 = 0.89$.

We hope you chose metamodel Alpha! Metamodel Charlie looks very bad—we've exaggerated this to make a point, but there should be no reason you'd want to include 600 additional terms to a regression metamodel to "explain" another 4% of the variability in the data, even if the terms all qualify as statistically significant. It would be difficult to even write the resulting model down on a page in any way that would be meaningful to a reader. You might think you could avoid this trap by using the adjusted R^2 since it will penalize you for adding extraneous terms to the model even if R^2 increases, but that is still not necessarily the "best" criteria to use for picking a final metamodel. Metamodel Bravo lies in between, but the 1% additional explanatory power doesn't seem worth requiring twice as many terms.

There are several issues that can effect our perspective and interpretation of "significance" when it comes to the individual terms in metamodels. These differ based on whether we are dealing with observational data collected in an ad hoc manner, or inferential data that arise from a designed experiment.

- For metamodeling with observational data:
  - Metamodel p-values are typically reported for individual tests, even when you are actually performing multiple tests. For example, multiple regression metamodels report p-values for each regression coefficient $\widehat{\beta}$. Unless you adjust the traditional $\alpha = 0.05$ significance threshold for these tests, you are not guarding against false positives appropriately. This highlights a serious flaw in many data mining studies, where spurious correlations are virtually certain to arise due to the sheer number of potential explanatory variables in observational datasets.
  - p-values are typically computed based on assumptions of i.i.d., normally-distributed errors. The p-values may be inaccurate if any of those assumptions are incorrect. With small datasets, in particular, and a strict threshold criterion (such as p-value < 0.05) for including or excluding a term from the metamodel, the structure of the fitted metamodel itself can also be quite sensitive to violations of these assumptions, the sample size, or presence of a high-leverage value of x.
- For metamodeling in a data farming context:
  - We often eschew α thresholds for the following reasons:
    (i) There are usually more than enough d.f. to claim statistical significance for even minuscule effects. When we simplify metamodels by removing terms with low p-values but little practical importance, we are implicitly using a smaller α value to assert statistical significance.
    (ii) The metamodels are based on carefully constructed designs that reduce spurious correlations between factors and responses.
    (iii) The number of potential simultaneous tests is bounded due to the structure of the study, specifically the number of factors and the type of metamodel,

in contrast to the essentially unlimited fishing expedition that is the basis of many data mining studies.

- Data farming metamodels are based on inferential big data rather than observational big data, so there is no ambiguity between cause and effect.
- For large data farming studies, practical importance dominates, our designs guard against unusual factor combinations, and so the metamodel selection process is robust to departures from the traditional regression assumptions.

4.8.4 What about outliers?

With observational data you may be very interested in "cleaning" the data, which involves (i) identifying outliers and deciding whether to keep, modify, or discard them; and (ii) deciding what to do about missing data. With simulation and data farming, missing data and outliers are handled differently. If you use automated run control for data farming, missing data should not occur unless the simulation has bugs, you have misspecified the run control requirements, or the output correctly reports a valid missing result due to censored data. We discuss this in more detail in Section 5.1 and Section 5.11. Outliers should not be removed when constructing metamodels in a data farming study. Instead, they should be investigated thoroughly as indications that either your model or your understanding is in need of adjustment. Unusual results in so-called outliers or outlying dps can lead to important and interesting insights, or reveal bugs in the simulation code. We discuss outliers and unusual dps in more detail in Section 5.3.

4.8.5 Semi-automated metamodel selection

Identifying reasonable metamodels can be assisted by automated methods. However, at this point in time it is still important to include the analyst's judgment in the metamodeling.

Multiple regression metamodels

Stepwise regression can be used to quickly arrive at a reasonable polynomial metamodel. Because our experience with simulation studies shows that interactions and nonlinearities are pervasive in simulation, we recommend beginning by allowing interactions and quadratic terms. We recommend some kind of forward or mixed stepwise process to get started. Backward stepwise selection can only be used if there are enough d.f. and the design is capable of fitting a full second-order metamodel. You can use backward stepwise selection with FBDs or CCDs, but not with NOLHs and NOABs and many other types of space-filling designs. However, as we discussed in Section 4.8.3, these automated methods may still result in metamodels that include terms of high statistical significance but low practical importance. For example, consider two metamodels fit to data from `TheOracle.rb` using `big_project_y.rb`

using `TheOracle.rb`—an instance where we happen to know the underlying ground truth. Partial information from JMP regarding two metamodels appears in Figure 4.18. The terms are sorted in decreasing statistical significance (i.e., decreasing $|t$-value$|$ and increasing p-value), producing a "tornado plot" that displays the relative impact visually. Figure 4.18a shows the results of using stepwise regression on a summarized 129-dp dataset based on 40 replications of a 129-dp NOLH, where the metamodel selected minimizes the AICc (corrected Akaike Information Criterion). This metamodel has 33 terms involving 15 of the 20 factors, and yields $R^2 = 0.996$. Figure 4.18b shows the results after manual simplification; in this case a metamodel involving only 5 factors and 10 terms still yields a very high $R^2 = 0.946$.

Term	Prob>|t|
x8	<.0001*
x14	<.0001*
x2	<.0001*
x4	<.0001*
x1	<.0001*
(x14-0.5)*(x14-0.5)	<.0001*
x3	<.0001*
x13	<.0001*
(x1-0.5)*(x2-0.5)	<.0001*
(x4-0.5)*(x14-0.5)	<.0001*
(x8-0.5)*(x8-0.5)	<.0001*
x11	<.0001*
(x3-0.5)*(x3-0.5)	<.0001*
x9	<.0001*
x7	<.0001*
x17	<.0001*
(x1-0.5)*(x15-0.5)	<.0001*
x15	<.0001*
(x2-0.5)*(x2-0.5)	<.0001*
(x9-0.5)*(x20-0.5)	0.0004*
(x3-0.5)*(x20-0.5)	0.0009*
(x11-0.5)*(x14-0.5)	0.0015*
x18	0.0052*
x20	0.0052*
(x1-0.5)*(x13-0.5)	0.0057*
(x4-0.5)*(x4-0.5)	0.0074*
(x1-0.5)*(x18-0.5)	0.0126*
x10	0.0141*
(x7-0.5)*(x20-0.5)	0.0144*
(x2-0.5)*(x4-0.5)	0.0319*
(x2-0.5)*(x7-0.5)	0.0557
(x3-0.5)*(x11-0.5)	0.1013

(a) Initial stepwise metamodel.

Term	Estimate	Std Error	t Ratio	Prob>|t|
x8	948.302	34.92438	27.15	<.0001*
x14	901.32394	34.92419	25.81	<.0001*
x2	490.24827	34.92429	14.04	<.0001*
x4	-479.4854	34.92401	-13.73	<.0001*
(x14-0.5)*(x14-0.5)	1526.152	144.0275	10.60	<.0001*
x1	347.6692	34.92402	9.96	<.0001*
(x4-0.5)*(x14-0.5)	-1010.997	131.4132	-7.69	<.0001*
(x8-0.5)*(x8-0.5)	798.39108	143.523	5.56	<.0001*
(x1-0.5)*(x2-0.5)	613.46227	110.9678	5.53	<.0001*
(x2-0.5)*(x2-0.5)	267.71129	134.5381	1.99	0.0489*

(b) Stepwise metamodel after further refinement.

Figure 4.18: Influential metamodel terms.

We remark that the interaction terms in Figure 4.18 are written as $(x_i - c_i)(x_j - c_j)$. This indicates the individual factor levels are centered by subtracting their column means for computational purposes before fitting the interaction effects (and similarly for quadratics). Centering also orthogonalizes the columns in the analysis matrix, making interpretation easier since the coefficients do not change numerically based on inclusion or exclusion of other metamodel terms. However, we will refer to these as simply the $x_i x_j$ interaction when talking about the terms included in the metamodel (and similarly for quadratics). Note also that many of the terms removed from the metamodel in Figure 4.18a are highly statistically significant, with p-values less that 0.0001.

Partition tree metamodels

One automated method for partition trees is to simply make 10 splits, and then calculate the relative impact (or influence) associated with each of the factors as proposed by Marlow et al. (2019). The relative impact is calculated by multiplying two numbers between 0 and 1: the R^2 for the tree, and the column contribution for the factor.

The relative impact of factor i can be categorized according to the largest inverse power of 2 it exceeds. Equivalently, the categorization is determined by the first occurrence in the following list:

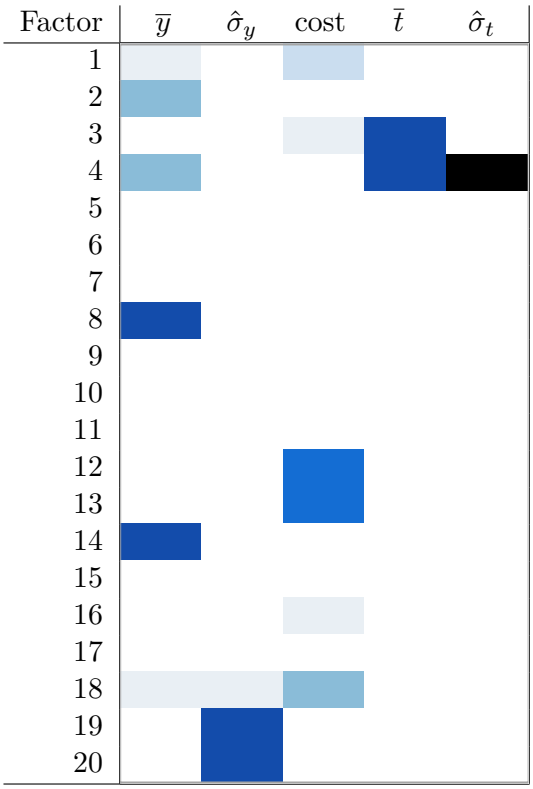
- VVS: very, very strong (≥ 0.50);
- VS: very strong (≥ 0.25);
- S: strong (≥ 0.125);
- M: moderate (≥ 0.0625);
- W: weak (≥ 0.03125); and
- VW: very weak (≥ 0.015625).

Any term failing to fall in one of these categories, or not included in any splits, is considered to be of no practical importance.

Of course, if you are seeking the "best" partition tree, then it may not involve 10 splits. Sometimes a 10-split tree may benefit from pruning, while at other times a 10-split tree still has room for improvement. However, the 10-split automated method has advantages. First, for any large-scale experiment, there will be enough dps to support making a 10-split tree. The tree will stop with fewer splits only if no further improvement is possible, e.g., if the first two splits for a binary Y has one leaf containing all the 0 values and another containing all the 1 values. That is not the case for all measures—if you tried splitting until you achieved a sufficiently high R^2, that will not be achievable for the raw data from a stochastic simulation experiment, and may not be achievable unless the minimum leaf size is set to 1.0 for summarized data; either case is problematic. Second, the 10-split approach naturally focuses attention on the most influential set of factors from the data: this is automatically prioritizing potential interventions a decision maker might take to achieve a good result. There can be at most 2 VVS, or at most 4 VS, or at most 8 S factors for any particular response. Finally, a 10-split approach is particularly useful when you are looking at a large numbers of responses and interested in seeing similarities and differences in the associated sets of influential factors. For example, `TheOracle.rb` can be used to simulate several responses related to the completion of a big project—specifically, the performance y, the *cost*, and the time t. Figure 4.19 shows the results when fitting 10-split trees to the summarized data, along with the relative impacts of the factors in both tabular and graphical form.

Factor	$\overline{y}$	$\hat{\sigma}_y$	cost	$\overline{t}$	$\hat{\sigma}_t$
1	VW		W		
2	M				
3			VW	VS	
4	M			VS	VVS
5					
6					
7					
8	VS				
9					
10					
11					
12			S		
13			S		
14	VS				
15					
16			VW		
17					
18	VW	VW	M		
19		VS			
20		VS			

(a) Labeled categories.



(b) Visual categories, where darker colors indicate higher contributions.

Figure 4.19: Relative factor contributions for various responses from an experiment involving `TheOracle.rb` simulation, after performing 10 splits for partition trees on the summarized output file.

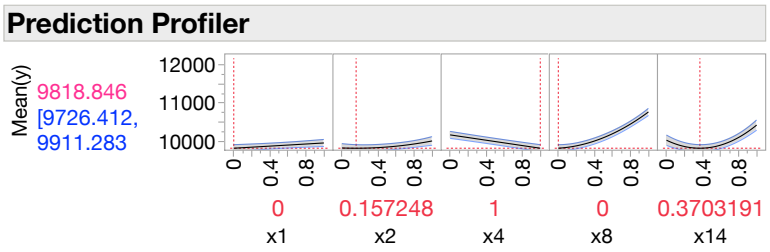
Use of multiple metamodels

It is often beneficial to fit more than one type of metamodel during the analysis process. For quantitative responses, we recommend fitting stepwise regression (with manual simplification) based on second-order polynomial terms, and a 10-split partition tree for both the mean and the standard deviation of the response of interest. If one has much better explanatory power than the other, use it! If both agree on which factors are the most important, and have similar explanatory power, then the choice of which to further refine, whether to create some sort of hybrid metamodel, or whether to explore other types of metamodels, is up to you as the analyst.

Confirmation runs

Don't forget that you have the luxury of being able to confirm any metamodel-based insights before you actually make recommendations. This is often not the case for those doing physical experimentation. For example, suppose you are interested in low response values arising from

the metamodel of Figure 4.18b. In JMP, the "prediction profiler" can be used interactively to suggest a good configuration as in Figure 4.20a. You can then make 40 runs at a single new dp corresponding to $x_2 = 0.157248$, $x_4 = 1$, $x_{14} = 0.3703101$, and all other $x_i = 0$, which is not one of your dps. Based on the 40 new runs you find the observed $\overline{y} = 9826$ falls handily within the 95% confidence interval from the profiler. The best of the original dps had a sample mean of 9965, so this new dp represents a 9% improvement over the centerpoint of the design matrix (all $x_i = 0.5$), and a modest improvement of 3.4% over the best of the original dps.



(a) Prediction profiler shows factor settings associated with a low response.

Summary Statistics

Mean	9826.56
Std Dev	91.85
Std Err Mean	14.52
Upper 95% Mean	9855.939
Lower 95% Mean	9797.187
N	40
N Missing	0

(b) Statistics from confirmation runs at new factor settings.

Figure 4.20: Relative factor contributions for various responses from an experiment involving `TheOracle.rb` simulation, after performing 10 splits for partition trees on the summarized output file.

Chapter 5

Top 10 Analysis Questions and Visualization Tips

Key Concepts

- To gain useful insights from data farming experiments, you must ask and answer questions that stakeholders and decision makers care about.
- Good statistical analysis software and visualization tools facilitate your exploration efforts. Interactive tools are particularly powerful.
- Analysis is a journey. Be prepared to modify your plans—and perhaps even your destination—based on what you discover along the way.

In this Chapter, we describe the Naval Postgraduate School SEED Center's "top 10 list" of analysis questions for data farming studies. We also provide examples of some visualizations that have not appeared in earlier chapters. These questions have been assembled based on decades of experience with data farming, but they are intended to kick-start your data exploration, not to stifle it.

In many cases, the data for the examples below were generated from one of two experiments involving reference simulations. We provide these datasets so you can reproduce the corresponding graphs below, or create other types of graphs and metamodels or other types of analysis using the identical data. Alternatively, we provide the description of the experiments so you can either generate similar data using the same designs, or generate new data after modifying factor ranges or the type of design. Note that we have used color-blind friendly palettes for most of the graphics, which requires some extra effort but may not be necessary for your own exploratory analysis.

- The file `big_project.csv` contains consolidated results from running 40 replications of `TheOracle.rb` with three separate models: `big_project_y.rb`, `big_project_cost.rb`,

and `big_project_time.rb`. This is a stylized example of a system with multiple responses (performance `y`, `cost`, and completion `time`). The design is a single stack of a 129-dp NOLH obtained using the command `generate_design.rb -d nolh` with all factor ranges between 0 and 1. The summary data file is `big_project_summ.csv`.

- The file `epidemic.csv` contains results from running 50 replications of `epidemic.rb`. The design is one stack of a 37-dp FBD crossed with a four-level factor for different transmission ratio reductions. We have added columns for two other potential responses as described in Section 5.1. The summary data file is `epidemic_summ.csv`.

We presume that before moving on to the top 10 analysis questions, you have a firm grasp of how to generate the designs to use as input data, and how to run the model using that design as input. If you're unsure, you should practice doing so and check that there are no surprises in your generated design file or your output. For instance, the number of rows in your raw output dataset should equal the product of n_{dp} and n_{reps}. If your job execution was cut short by a power surge or a limit on CPU time, just pick up where you left off.

Look at histograms, boxplots, or summary statistics for the factors to confirm that you have the ranges you intended. If not, follow up to see whether (i) you fat-fingered an entry when generating the design, (ii) the factors in your design appear in a different order than the simulation input requirements; (iii) one or more required inputs are missing; or (iv) your simulation was expecting input in a different format. An example of (iv) that we once encountered involved entering a probability (e.g., in the range 0.10 to 0.25) when the simulation was written to use an integer percentage (e.g., in the range 10 to 25) and subsequently rounded all factor levels to zero.

Bear in mind that a histogram's appearance is affected by the range boundaries used to categorize the bins. Also note that histograms of factors levels will look different depending on the design. For example, histograms have just three few levels for (non-rotatable) CCDs, look fairly uniform for NOLHs and NOABs, and are more U-shaped for FBDs and FBSs. Descriptive statistics for qualitative factors are frequency distributions rather than histograms. For example, Figure 5.1 shows a histogram, boxplot, and quantiles of the first of 20 factors from the experiment of `big_project.csv` (the other factors have identical pictures), and Figure 5.2 shows these for the 5 factors from the experiment of `epidemic.csv`.

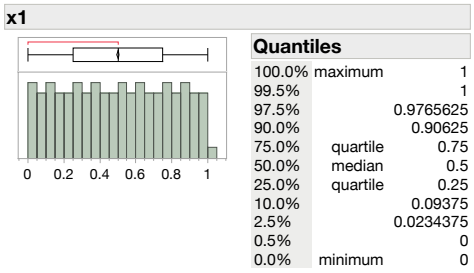


Figure 5.1: Histograms of factor levels for the big project simulation experiment.

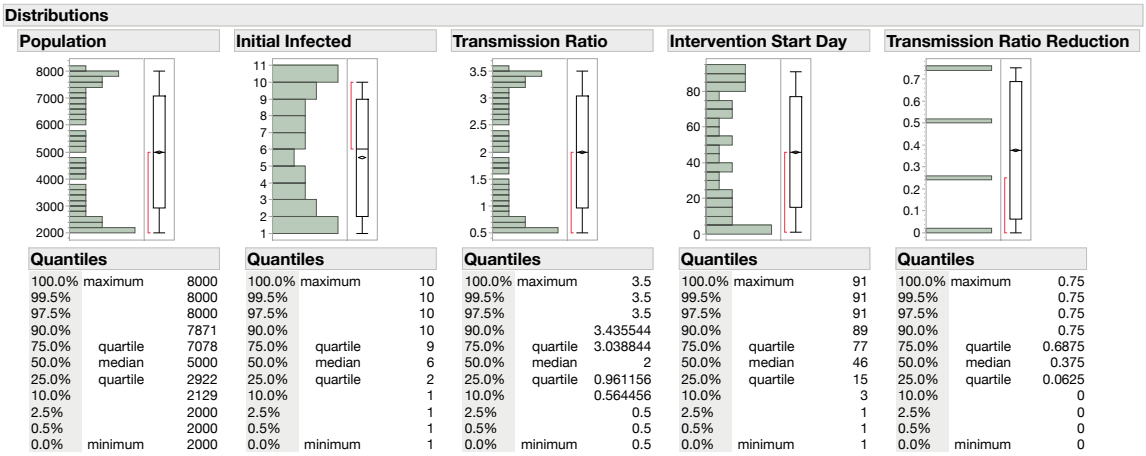


Figure 5.2: Histograms of factor levels for the epidemic experiment.

5.1 Q1: What is the overall spread of the responses?

Before diving deeply into analysis, make sure your primary responses have reasonable values and also enough variation that analysis will be worthwhile. Questions 1 and 2 use simple descriptive statistics, histograms, and boxplots to address this. Interactive data exploration tools are particularly useful.

When looking at histograms or boxplots of the responses, it's important to remember that you are mixing data from many different dps and it does not make sense to try fitting a distribution to the responses—or even thinking about the results distributionally. Instead, you are looking at the range of outputs to verify that (i) they are complete—you have output data for each run; (ii) their values seem plausible; and (iii) there is enough variation in those response measures of interest that it is worthwhile attempting to explore them further.

If your dataset fails to meet criterion (i) or (ii), you must track down the problem in the design or in the simulation model and then regrow your data before moving on. A quick example follows. If you run `TheOracle.rb` simulation with the `big_project_y.rb` model using the design specifications in `big_bad_design.yaml`, you will notice that some of the responses are missing. (In this situation, the warning "Input out of bounds [0, 5]" prints to the screen during run execution, but you may not have seen this if you made runs on a cluster.) A partition tree with a single split perfectly isolates these missing results ($R^2 = 1$) and shows they correspond to dps with $x_{16} < 0.003125$. If x_6 should be a non-negative number, then the design is bad: change its minimum in the `big_bad_design.yaml` file and regenerate the design with `generate_design.rb -i big_bad_design.yaml` to get a usable design. If x_6 could meaningfully take on small negative values, then change the ranges in `big_project_y.rb` from [0,5] to (say) [-1,5] and rerun the dps that failed to execute initially. The use of analysis and visualization tools to assist with model debugging is discussed in more detail in Section 5.11.

If (iii) occurs, this is not necessarily a problem with your study—it simply means that the only insight that can be drawn about that particular response from this experiment is that, for practical purposes, it doesn't change. For example, Figure 5.3 has histograms of six potential responses after summarizing the `big_project.csv` results. The average cost ranges from \$21.30 to \$30.01, but the range of standard deviation values is less than one cent. This is why we did not include `Std Dev(cost)` when seeking relative factor contributions for various responses in Figure 4.19. Don't be fooled by the shape of the histogram, it is essential to look at the scale and think about what it means. As another example, in one study a researcher claimed they had visual proof that an orthogonal design was not orthogonal. When we looked at the scale for the histogram, all of the data fell between -1E-17 and 1E-17—the precision of double precision floating point arithmetic on a computer.

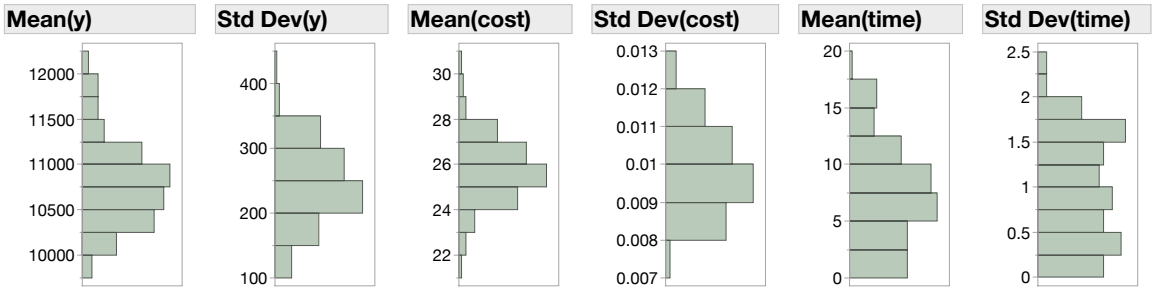


Figure 5.3: Histograms of several responses from the data, summarized by design point, for the big project simulation.

Keep in mind that the outputs from the simulation may not be the only responses you are interested in analyzing. We've already stated in Chapter 4 that it is important to consider a measure of central tendency (e.g., mean or median) and a measure of spread (e.g., standard deviation or interquartile range) for each response, and measures of spread arise when we summarize the raw data. We might also decide to add new response columns to our output data set. For example, the `epidemic.csv` file reports the duration of the epidemic, along with the number of the population infected or uninfected during this time span. We might be interested in adding new columns to the data, which represent two other potential performance measures. One is the proportion of the population ever infected, defined as `proportion_infected=Infected/Population`. This could be useful to stakeholders responsible for at-risk communities of different sizes. Another is a binary factor `Flare` defined to be 1 if `proportion_infected` ≥ 0.10 and 0 otherwise. This might be much more useful in discussions about risk than reporting the average number or average proportion infected.

Finally, we can sometimes get some interesting initial insights by looking at these simple graphs interactively. For example, Figure 5.4 is a snapshot of interactive histograms for the epidemic experiment with the two new responses columns. Note that due to page width limitations we are only showing one factor. In the statistics package we used, highlighting by clicking on one or more bins in one histogram highlights the selected observations in all other histograms or plots. We have highlighted the lowest level bin (0) for transmission rate reduc-

tion in the left-most histogram. Notice that the outcomes this highlights in Prop(Infected) are very near either 0 or 1, but nothing in between—a finding that is not directly evident from the histogram of Uninfected. We also see correspondingly more flares than fizzles in the Flare histogram.

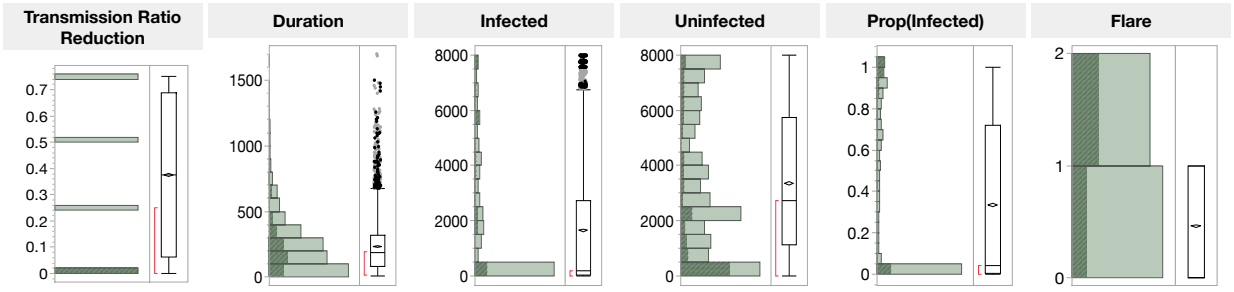


Figure 5.4: Histograms of one factor (transmission ratio reduction) and several responses from the epidemic experiment. Highlighted portions of the bars correspond to runs with `Transmission Ratio Reduction` equal to 0.0.

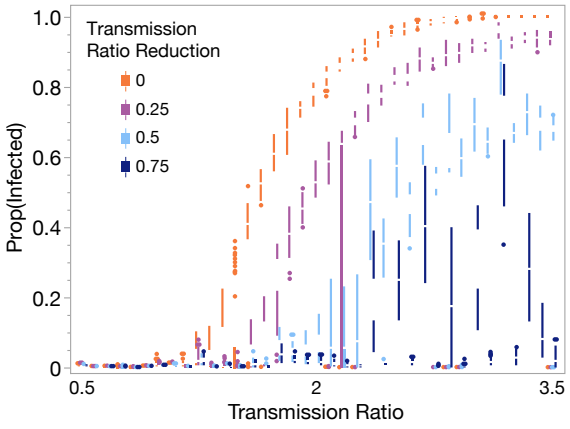
5.2 Q2: What's the distribution across replications for each design point?

A second use of histograms and boxplots involves looking at them by dp. In this case, we are interested in the distributional behavior. Are the results symmetric or skewed? Are they unimodal or multimodal? Do they exhibit low variability or high variability? Finally, are they similar across dps or not?

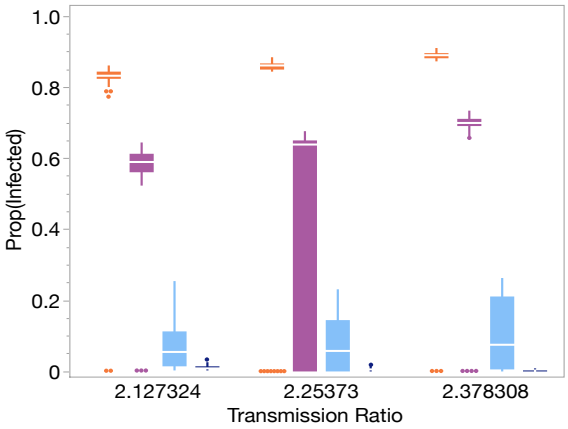
Answering these questions may require you to add a column for the design point, if the output does not already contain some type of identifier. We have already done this for the reference datasets `big_project.csv` and `epidemic.csv`. Some designs, such as FBDs, have only one dp associated with any particular factor level if there's not much rounding and only one stack.

End of run or aggregate behavior

Consider Figure 5.5 that shows some results from the epidemic experiment. Figure 5.5a shows box plots for each dp, i.e., the only variation in a particular box plot occurs across the 50 replications. The dps are ordered by increasing values of the transmission ratio, then by increasing transmission rate reduction, and some distinct behaviors are already evident. There are dramatic differences in both the mean and the variability of `Prop(Infected)` across the dps. Many of the dps yield proportions that are always very close to zero, some only yield proportions very near 1.0, and others exhibit a great deal of variability in their outcomes, as evidenced by long boxes or outliers far from the whiskers in the box-and-whisker plots. Figure 5.5b zooms in on a portion of the graph containing the dp with the largest box.



(a) Box plots of Prop(Infected) by dp, ordered by transmission ratio and color-coded by the transmission ratio reduction.



(b) A closer look at boxplots for several of the dps with highly variable results.

(c) Histograms for dps with a transmission ratio of 2.25373 and four transmission ratio reductions highlights the bimodality in many of the results.

Figure 5.5: Box plots and histograms of Prop(Infected) from the epidemic experiment.

This perhaps makes it easier to see that all twelve of these dps had some runs where the epidemic fizzled. For the lower two transmission ratio reductions, the separations between the fizzles and the flares are quite large, showing that these distributions are bimodal. Figure 5.5c reveals that this bimodality is also present for some dps without this distinct separation.

Ordering or partially ordering the dps by one or more factors, as we did in Figure 5.5, can provide hints about the nature or structure of the underlying relationship. However, it is not necessary to order the dps to address Q2. Figure 5.6 shows boxplots of the duration vs. dp for the epidemic experiment. From this it is evident that the dps are quite different in terms of both their centers and their spreads. Some of the boxplots appear relatively symmetric, others are highly skewed, and some appear to have bimodal behavior. This may be easier to identify by zooming in on a portion of the dps, as in Figure 5.7, or by examining histograms. Public health officials, medical professionals, and the public should all be concerned to note that in certain conditions some of the outbreaks are active for 2–3 years. It would be worthwhile to construct two metamodels related to duration: one for its average, and another for its standard deviation, to communicate the degree of risk involved.

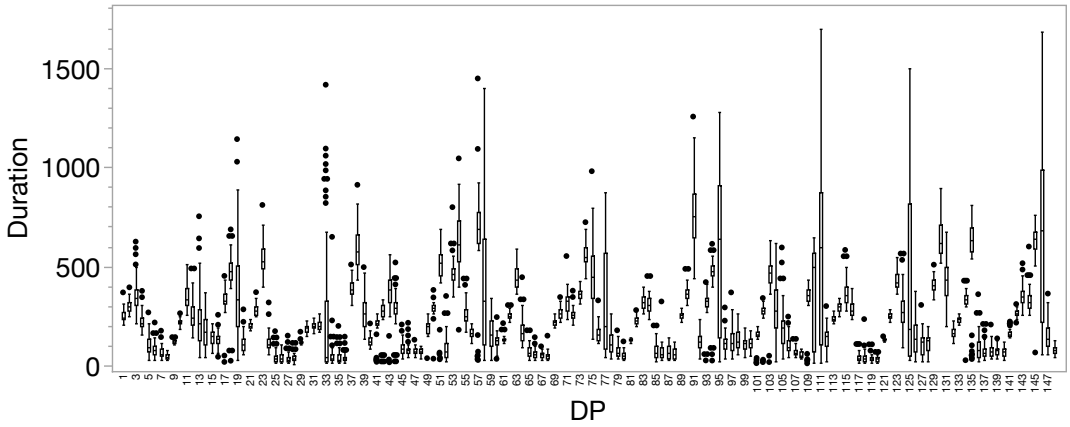


Figure 5.6: Boxplots of duration versus dp for the epidemic experiment.

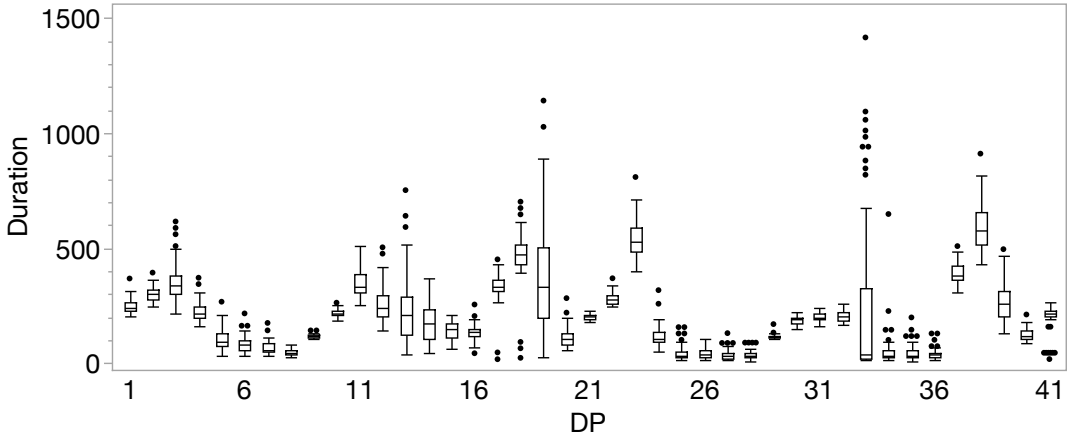
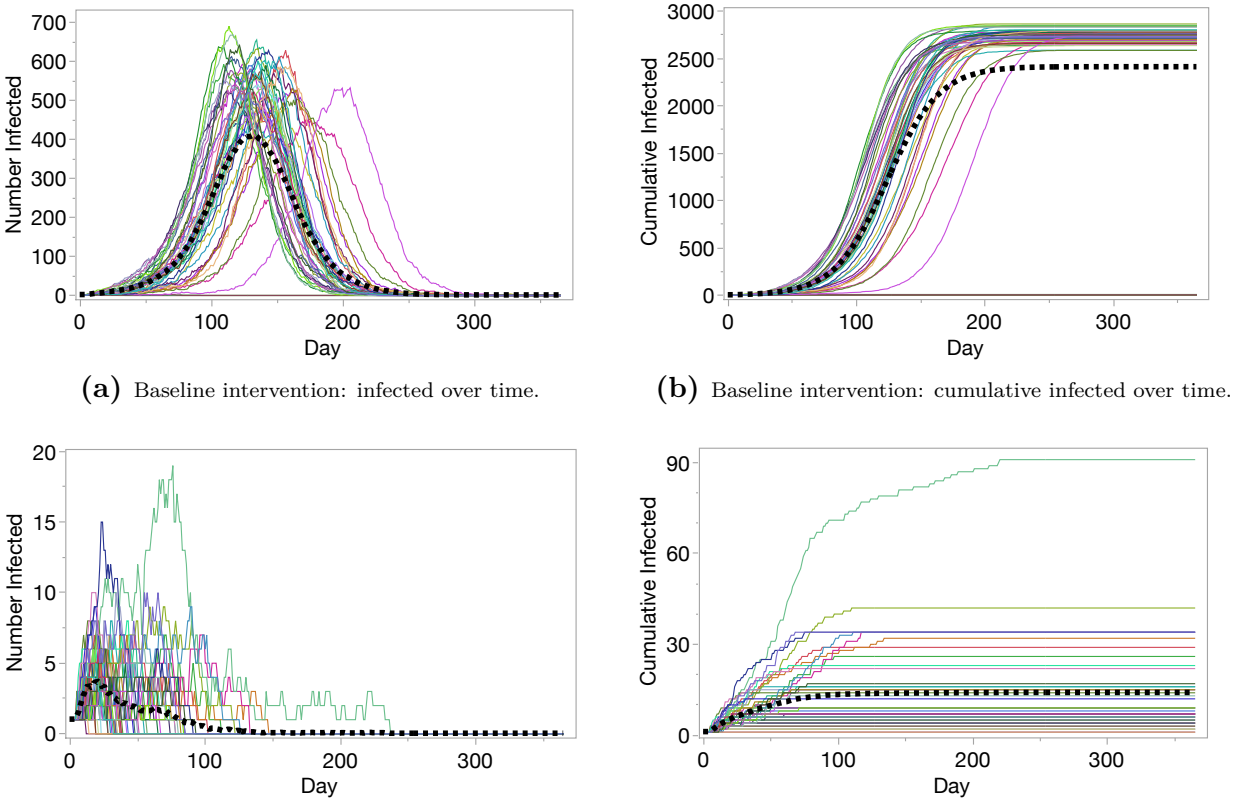


Figure 5.7: Boxplots of duration versus dp for the first 41 dps from the epidemic experiment.

Behavior over time

Graphs such as those in Figure 5.5 consider end-of-run statistics for each replication. Additional insights can be gained for dynamic systems by looking at response behavior over time. For example, results from 50 replications for two dps of a related epidemic model appear in Figure 5.8. Figures 5.8a and 5.8b each show the daily number and cumulative number of infected individuals for a baseline intervention scenario. Figures 5.8c and 5.8d do the same for a more aggressive intervention scenario, where the intervention used is both more effective and applied earlier in the epidemic.

In all subplots of Figure 5.8, the heavy dotted lines are the averages across all replications in the respective graph. Note that all graphs exhibit fizzles, where the number infected drops to zero relatively early in the year, and the cumulative number of infected remains low. Since the fizzle epidemic trajectories have a high degree of overlap, their prevalence is more easily seen by their effect on the average end-of-year behavior. An interesting illustration of this is that



(a) Baseline intervention: infected over time.

(b) Baseline intervention: cumulative infected over time.

(c) Aggressive intervention: infected over time.

(d) Aggressive intervention: cumulative infected over time.

Figure 5.8: Epidemic progress over time for two different intervention strategies used for a population of size 4000. Solid lines show epidemic trajectories for each of 50 replications, while heavy dotted lines denote the averages across these replications. For the baseline intervention, the average number infected during the first year is 2406, or 60% of the population. The more aggressive intervention yields an average of 14 infections during the first year, or less than 1% of the population.

the heavy dotted line is separated from the individual trajectories in Figure 5.8b, indicating that it falls between the two modes of the bimodal distribution. The daily trajectories also reveal the extremely high degree of variability in the timing and height of the peaks for the individual epidemic curves. Different stakeholders might use this information in different ways once they become aware of this phenomenon. Hospital administrators might be very concerned about the peak value, while public health officials might find it prudent to emphasize that different communities could experience very different trajectories as they seek to educate their constituents about local risk.

5.3 Q3: Are there outliers or unusual design points?

As discussed in Section 4.8.4, so-called outliers or unusual dps may be very interesting points in a datafarming dataset and should not be discarded. Before presenting examples, a bit more discussion is in order. First, the definition of what constitutes an outlier is often taken as a

point outside of roughly 3 standard deviations from the mean for normally-distributed data (e.g., outliers in residuals after fitting a regression metamodel when the classical assumptions hold true). The nonparametric version used for boxplots is that an outlier is a point outside the whiskers of a boxplot. That corresponds to a point over 1.5×IQR from the top or bottom of the box, where IQR is the interquartile range—the length of the box. If the true distribution of the data is normal, either approach should result in a very small proportion (roughly 0.005) of the observations classified as outliers. If the data come from a skewed distribution, such as an exponential, then a larger portion of the observations will be classified this way.

In data farming studies where data are plentiful, a different perspective is needed. We can think of most of the points that are automatically tagged as outliers as, instead, simply observations coming from the tails of the distribution, or as evidence of skewness or other types of non-normality such as bimodality. For example, recall the histograms and boxplots for the epidemic experiment in Figure 5.4. According to the boxplots, there are 265 outliers for the epidemic duration and 345 outliers for the number of infected. However, the histograms show that these points are not so unusual. In contrast, a true outlier has a value that truly surprises us in the context of the bulk of the data. This could be due to a variety of reasons: a very rare but legitimate event; a potential bug in the simulation model; a potential explanatory factor excluded from the analysis; or a shift out of a steady-state system. Consider the following example from a data farming study of a pilot training facility simulation. The primary response is the overall time required to complete pilot training, a process that takes many months. Figure 5.9 provides the histogram, boxplot, and summary statistics for the experiment as a whole. The small number of outliers corresponding to training days of over 500 days are widely separated from all the rest of the data. Our focus here is on using available tools to see that something odd is going on. We will discuss possible approaches to deal with it in Section 5.11.

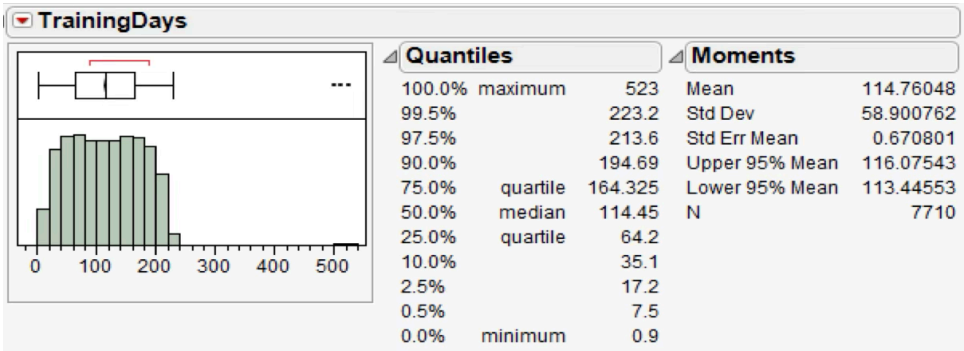


Figure 5.9: Histogram, boxplot, and summary statistics from a pilot training facility study.

Very rare events are sometimes referred to as black swans. In the infancy of data farming before more sophisticated DOE was incorporated, the most common infrastructure was set up to perform gridded full factorial experiments involving a handful of factors. Simple agent-based models were used to simulate some of the intangibles of combat (such as trust, training,

human response to changes in their local environment) that were difficult to incorporate in detailed, largely-deterministic, physics-based models of the time. The search for outliers was explicit: if there was only one time in 100,000 replications that the blue side could prevail, the stated hope was that by finding that one time, some potential reasons for success could be identified and then further tested. For example, if the favorable run was the only one where the blue side reached a particular location before a particular time, the simulation could be modified so the blue agents would be given guidance to move toward that location early. Additional runs could confirm or refute the idea that such guidance would improve their chance of success. Still, this approach shows that outliers can be very important.

Let's now consider unusual design points. Since design points exist by construction, these will not be outliers in terms of the factor level combinations. Instead, we consider them to be noteworthy or attention-grabbing if their behavior is very different from the patterns observed for other responses.

End of run or aggregate behavior

First, consider identifying outlying dps from end-of-run statistics. Boxplots or histograms of the raw data (i.e., one observation per replication) can be used to identify unusual dps. For example, see the notional data in Figure 5.10. We have highlighted several dps of potential interest: dp 7 is quite a bit lower than all others; dp 10 has very low variability; and dp 12 has very high variability. We will discuss dp 22 later. It is important to look at the all of the dps when deciding whether or not to classify a dp as an outlier. After all, for the epidemic experiment results in Figure 5.6 and Figure 5.7 the boxplots are much more dissimilar than those in Figure 5.10, so an outlying dp would have to appear even more extreme to stand out from the rest. To summarize the concept, the yardstick for judging outliers is the rest of the data.

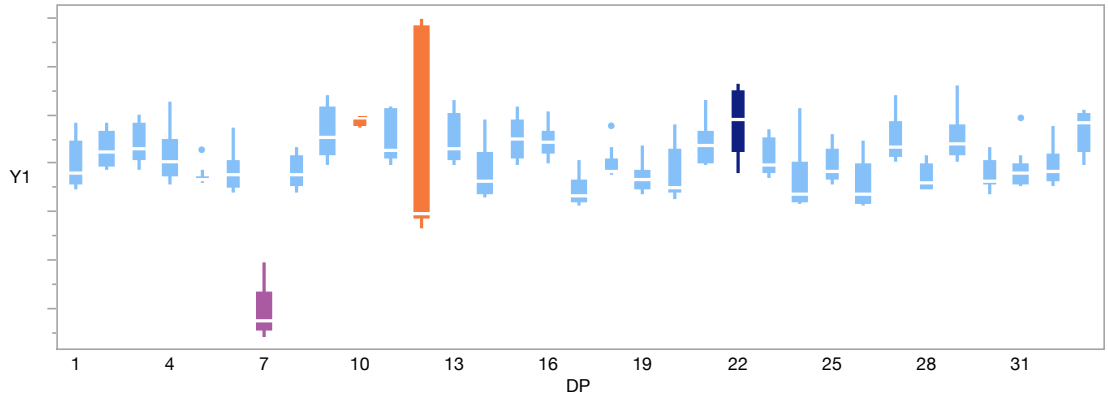


Figure 5.10: Boxplots of Y vs. design point for 10 replications of data from a notional experiment. Four boxplots are highlighted. Design point 7 yields much lower values than any others; dps 10 and 12 exhibit unusual variability, either very low or very high. The fourth highlighted boxplot (for dp 22) does not appear unusual in this plot.

While boxplots of the raw data can be good visual tools for identifying dps that are unusual for a single response, plots involving the summarized data might be better at revealing dps that differ in the case of two or more responses. Figure 5.11 is a scatter plot of the summarized data, by dp, for Y1 from Figure 5.10 and another response Y2; the color-coding from Figure 5.10 is also used. Figure 5.11 shows that dp 7 is unusual in terms of both Y1 and Y2, but does not seem out of kilter with a line projected through the major axis of the majority of the data. Neither dp 10 nor 12 appear unusual in this plot because it is a plot of means, not variances. However, dp 22 does seem unusual: it has an average Y1 value quite a bit higher than might be anticipated from the the rest of the data.

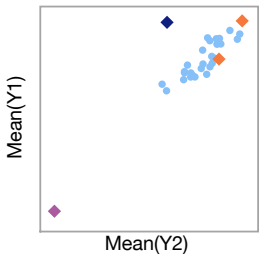


Figure 5.11: Scatterplot of two responses from a notional experiment, using the color-coding and Y1 data from Figure 5.10 along with diamond-shaped markers for four design points of interest.

Behavior over time

Just as it may be interesting to explore trace data for individual replications, it may also be interesting to explore how average trajectories differ by dp. For example, the two dashed curves in Figure 5.8 look very different.

5.4 Q4: Are any responses related or correlated?

We assume that for most data farming studies, there are multiple responses of interest. Responses can be quantitative or qualitative, and these categories often require different techniques. Keep in mind that understanding the relationships among various responses is extremely important to stakeholders and decision-makers—particularly if tradeoffs are involved.

Quantitative responses

Relationships between quantitative responses can be summarized numerically by a correlation matrix. Suppose that a set of two or more quantitative responses have very strong pairwise correlations (either positive or negative). This would allow us to focus subsequent analysis effort on just one response from the set, knowing that it is a strong predictor for the others.

As an example, Figure 5.12 shows pairwise correlations between ten responses from the epidemic simulation experiment. We use abbreviated labels to conserve space: those without a subscript denote mean responses, while those a subscript of `s` denote standard deviations, i.e., `Dur` is the Mean(Duration) and `Dur_s` is the StdDev(Duration). The highlighted box at the intersection of `Inf_s` and `Un_s` tells us that the standard deviation of `Infected` and the standard deviation of `Uninfected` are perfectly correlated with each other. Consequently, there would be no need to spend precious analyst time in separately constructing metamodels of both. After building a metamodel of (say) `Inf_s`, simply specify the same terms to fit the metamodel for `Un_s`.

	Dur	Inf	Un	Pr	F	Dur_s	Inf_s	Un_s	Pr_s	F_s
Dur	1.0000	0.3494	-0.2108	0.3530	0.6041	0.5870	0.3911	0.3911	0.3980	0.5180
Inf	0.3494	1.0000	-0.5984	0.8664	0.7908	-0.1314	0.3809	0.3809	0.3119	0.0564
Un	-0.2108	-0.5984	1.0000	-0.7645	-0.6967	0.1942	-0.1603	-0.1603	-0.2697	-0.0572
Pr	0.3530	0.8664	-0.7645	1.0000	0.9111	-0.1715	0.3071	0.3071	0.3463	0.0505
F	0.6041	0.7908	-0.6967	0.9111	1.0000	-0.0098	0.3681	0.3681	0.4166	0.1761
Dur_s	0.5870	-0.1314	0.1942	-0.1715	-0.0098	1.0000	0.3668	0.3668	0.3673	0.7748
Inf_s	0.3911	0.3809	-0.1603	0.3071	0.3681	0.3668	1.0000	1.0000	0.9193	0.6518
Un_s	0.3911	0.3809	-0.1603	0.3071	0.3681	0.3668	1.0000	1.0000	0.9193	0.6518
Pr_s	0.3980	0.3119	-0.2697	0.3463	0.4166	0.3673	0.9193	0.9193	1.0000	0.7158
F_s	0.5180	0.0564	-0.0572	0.0505	0.1761	0.7748	0.6518	0.6518	0.7158	1.0000

Figure 5.12: Correlation matrix of relationships among ten responses for the epidemic experiment.

Scatterplots can supplement the use of correlation matrices to provide us with a visual representation of potential pairwise relationships. Consider the highlighted item in the far right column of each plot which shows the correlation between the mean and standard deviation of a flare (`F` and `F_s`, respectively). Although the correlation is relatively low at 0.1761, Figure 5.13 shows that the two responses have the classic parabolic shape of a quadratic relationship.

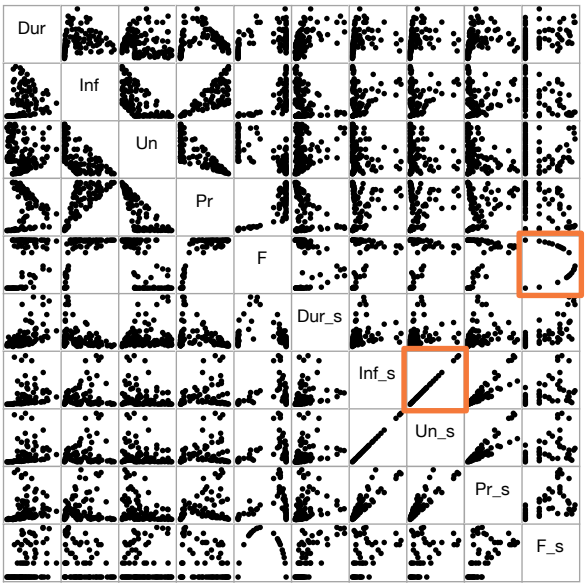
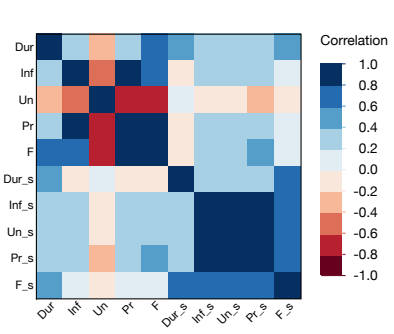


Figure 5.13: Scatterplot matrix of relationships among ten responses for the epidemic experiment.

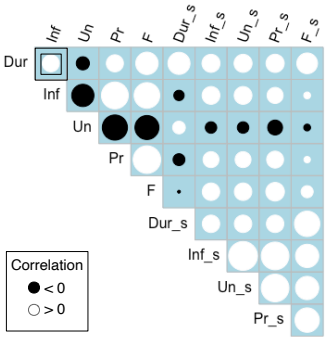
For this simple simulation, neither of these relationships should be surprising. `Flare` is a binary-valued response, so we know its standard deviation is a quadratic function of its mean. Similarly, since the sum of `Infected` and `Uninfected` equals the population size, and that size is fixed for any run, the standard deviations will be identical. However, in more complicated experiment settings, it may not be obvious *a priori* how closely different responses are related. In such situations, we could reuse the same metamodel terms for responses that were highly correlated, even if the correlation was not perfect.

Analysts should also consider whether correlations between pairs of performance metrics are positive or negative. If you're seeking system improvement, the sign provides a partial indication about whether factor adjustments that improve one will tend to be serendipitous (producing improvements in the other) or conflicting (worsening the other).

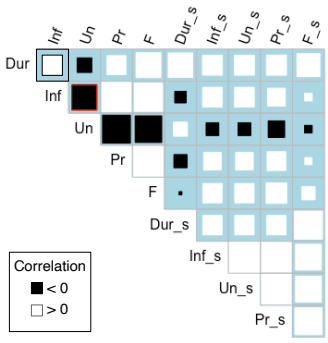
Analysts should also consider color-coded correlation plots. These are advantageous if the number of factors or responses is large and correlation matrices become unwieldy to look through. Figure 5.14 shows three such plots for the epidemic experiment. Standard color-coding for these types of plots is typically red-to-blue or blue-to-red as correlations range from -1 to 1, but are often user selectable as in the color map of correlations generated by JMP in Figure 5.14a. Unfortunately the available standard colors are often not distinguishable for all types of color-blindness, but their intensities reveal the absolute values of the correlations. Two color-blind friendlier alternatives are also shown. Both use black or white to denote the sign of the correlation, and the size of a shape to denote its strength. They give somewhat different visual impressions of the strengths since the diameter of a circle in Figure 5.14b is roughly the length of the side of the corresponding square in Figure 5.14c. Neither shape's area is directly proportional to correlation, which is unfortunate since the human eye judges magnitudes based on area. For example, the the numerical correlation between duration and infected in the upper-left corners is 0.35, the size of the corresponding circle is 31% of its box, and the size of the corresponding square is 40% of its box.



(a) Shaded boxes use standard coloring to represent sign, color intensity to represent strength.



(b) Color of circle represents sign, size of circle represents strength.

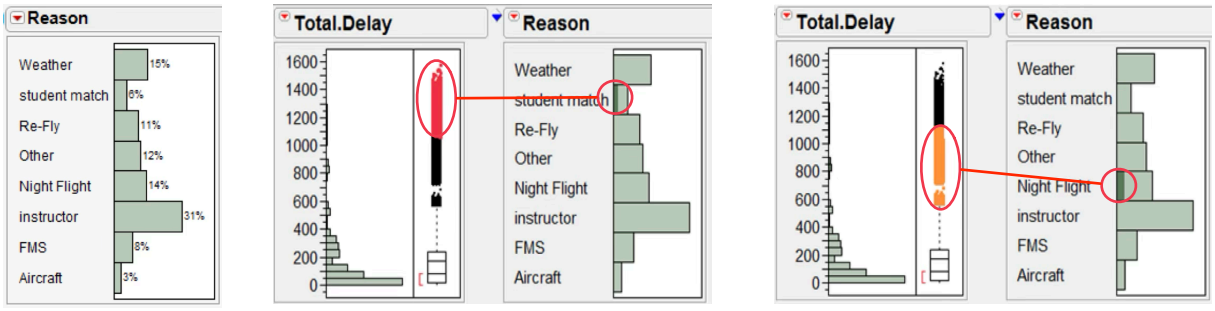


(c) Color of square represents sign, size of square represents strength.

Figure 5.14: Three correlation plots involving ten responses from the epidemic experiment.

Qualitative responses

When one or more responses are qualitative, then correlation cannot be used to capture the strength of the relationship. Contingency tables are one possibility, but graphs can often be very effective in revealing these relationships—particularly interactive graphs. As an example, consider a simulation of a training facility for pilots of high-performance jets. Figure 5.15 shows how histograms and boxplots can be used interactively to reveal information about their joint behavior. The primary response measures deal with the overall time required to complete pilot training, a process that takes many months. `Delay` refers to a delay in the graduation date projected when the student began flight school. However, the simulation generates several useful intermediate outputs, including delays when a student pilot has been unable to complete a practice flight as scheduled. Delays at any point in the schedule can propagate to subsequent tasks via rescheduling requirements. This would affect not only the primary student involved, but other students involved in team training exercises as well. Figure 5.15 shows how histograms and boxplots can be used interactively to reveal information about their joint behavior.



(a) Reason for delay.

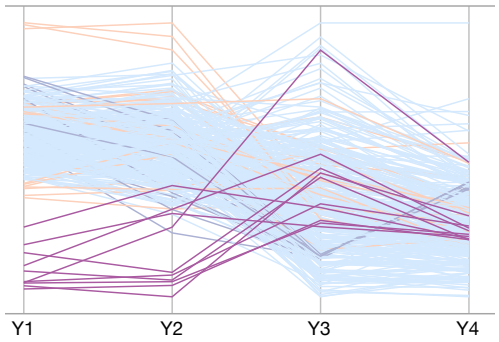
(b) The longest delays were associated with lack of a student match.

(c) Long delays are also associated with night flight certification.

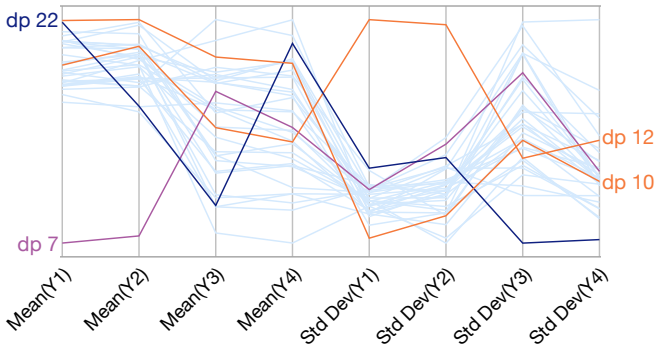
Figure 5.15: Delays occurring for student pilot training. Lack of an available instructor and bad weather were the most common reason for a delay, but did not result in the longest delays.

Parallel plots are another useful visualization technique. Each trace represents a single row in the dataset, connecting its values for different columns. The vertical axes are scaled between the maximum and minimum values for the respective columns. Figure 5.16 shows two parallel plots for the notional experiment of Figures 5.10 and 5.11. In Figure 5.16a, the four columns are the four responses Y1–Y4, and each trace is the result of a single simulation run. The traces for dp 7 are highlighted, and reveal the variability across the replications as well as the fact that they have low values for Y1 and Y2 compared to the bulk of the data. The plot in Figure 5.16b is from the summary data, so each trace represents the mean or standard deviation of a response measure across all replications for a single dp; we highlight all four "unusual" dps in this plot. Notice that dp 7 stands out as yielding low averages for both Y1 and Y2, dp 12 for its high standard deviations for Y1 and Y2. Dp 10 tends to yield high response means and low or fairly low response standard deviations. Most of the traces segments connecting Mean(Y3) and Mean(Y4) are relatively flat, indicating that these

two responses are positively correlated. (If two adjacent responses were strongly negatively correlated, we would see a mix of positive and negative slopes more like a bowtie.) Dp 22 behaves differently, with a steep rise.



(a) Four responses from the raw dataset, with traces for dp7 highlighted.



(b) Eight responses from the summary dataset, with traces highlight for four design points of interest.

Figure 5.16: Parallel plots for a notional experiment.

5.5 Q5: Which factors are most influential?

Note that questions 5, 6, and 7 all make extensive use of metamodels—sometimes directly, and sometimes to identify what types of 2- or 3-dimensional plots might be the most useful to construct. We already illustrated the basic approaches in Chapter 4—specifically, (i) using stepwise regression to fit a second-order polynomial metamodel and manually simplify it if warranted, and (ii) fitting a 10-split partition tree. For purposes of identifying influential factors, these metamodels can be constructed for a summarized data set, i.e., where you begin with the most interesting responses found in question 4. The relative impacts for factors in the big project experiment were shown in Figure 4.19, so we will not reproduce them here. The relative impacts for factors in the epidemic experiment appear in Figure 5.17. Note that for the epidemic experiment, every one of the factors has at least a moderate (M) relative impact for at least one of the responses. This is quite different from the results for the big project experiment, where of the 20 factors, eleven have at least a moderate impact on at least one response, one has only a very weak (VW) relative impact on a single response, but eight have no relative impact on any responses. The influential factors in the big project experiment is a smaller subset set of the total.

If a factor is deemed to be influential, there are many types of graphs that can be used to help reveal the nature of the overall relationship between that factor and the response. We also consider a factor to be influential if it is part of an influential interaction. Our experience is that many of the key insights arise from interactions and nonlinear relationships.

Term	Dur	Inf	Pr	F	Dur_s	Inf_s	Pr_s
Transmission Ratio Reduction	S	VS	VS	S	M		S
Transmission Ratio	VS	VS	VVS	VVS	S	VS	VS
Population		S			W		
Intervention Start Day			VW	M			M
Initial Infected					M	M	S

(a) Labeled categories.

Term	Dur	Inf	Pr	F	Dur_s	Inf_s	Pr_s
Transmission Ratio Reduction							
Transmission Ratio							
Population							
Intervention Start Day							
Initial Infected							

(b) Visual categories, where darker colors indicate greater impact.

Figure 5.17: Relative factor contributions for various responses from the epidemic experiment after performing 10 splits for partition trees on the summarized output file.

5.6 Q6: Which interactions are most influential?

As we mentioned in Section 3.2, in the multitude of systems we've studied, our experience is that interactions are almost always present and almost always interesting. This is why we recommend designs capable of revealing interactions, and allow interactions and quadratic effects as potential terms when we use stepwise regression for metamodeling. Interactions can be directly identified from the metamodel output: the tornado plot for the big project simulation in Figure 4.18 shows an important interaction between factor x4 and factor x14 (two factors that also happen to have important main and quadratic effects), as well as an interaction between x1 and x2. Tornado plots for the metamodel coefficients in the epidemic experiment show that the interaction between `Transmission Ratio` and `Transmission Ratio Reduction` is important for `Duration` and five of the other six responses in Figure 5.17. Interactions can also observed by looking more closely at partition trees and finding differences in split factors or split thresholds on different branches of the tree. For example, in Figure 5.18 we have highlighted splits involving the `Transmission Ratio` in purple, and those involving `Transmission Ratio Reduction` in light blue. Clearly, the left and right branches have different split patterns.

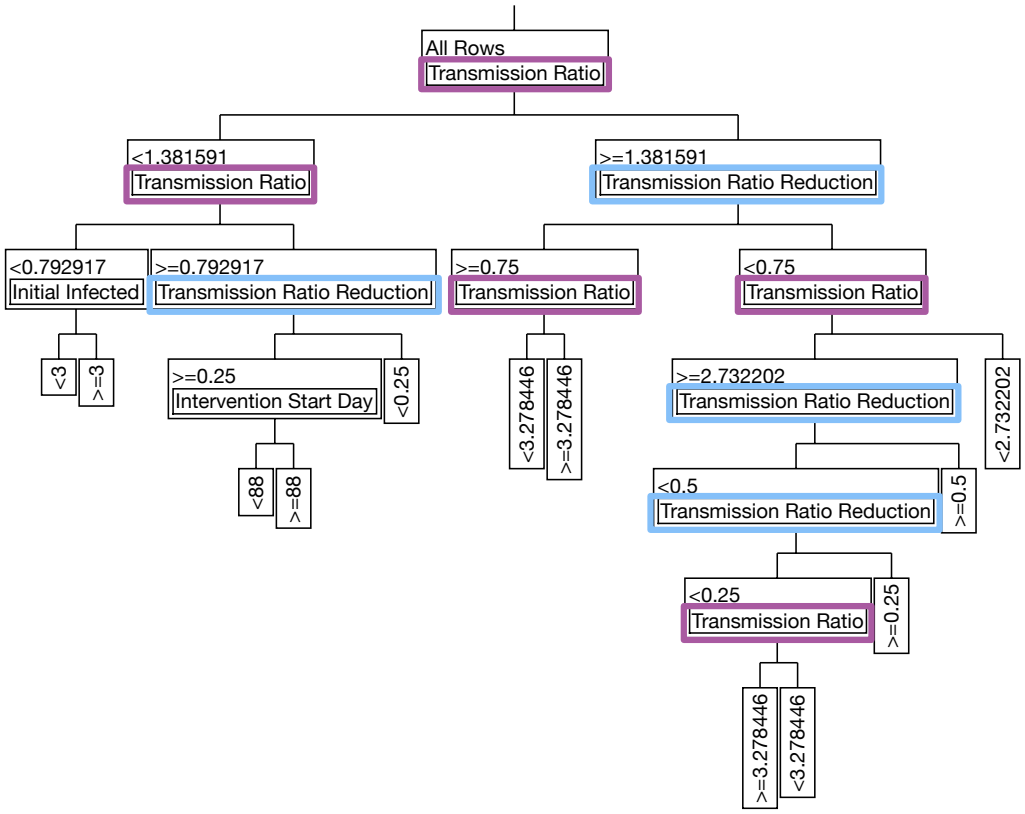
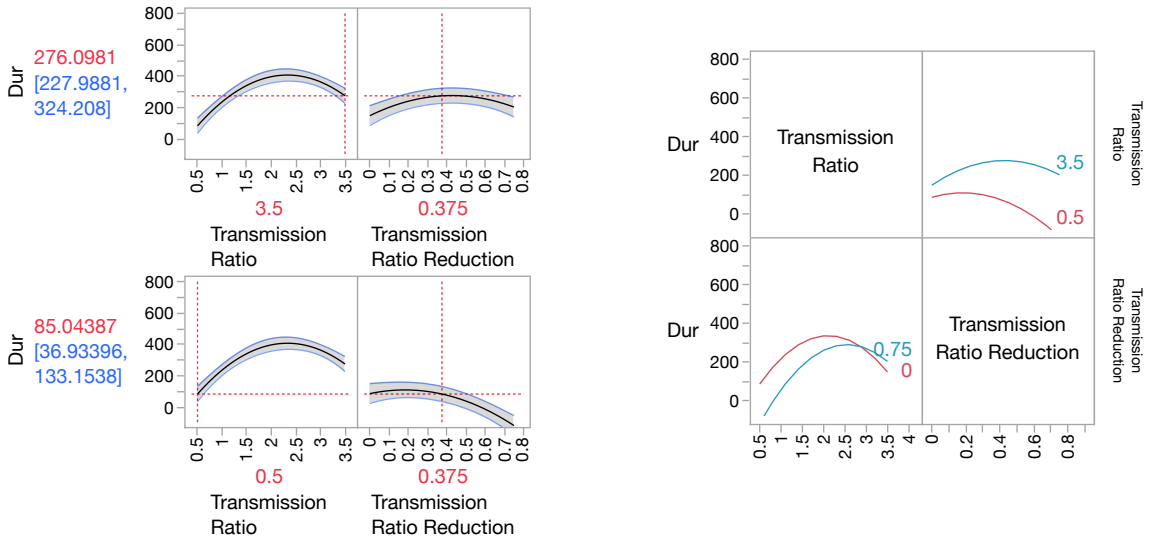


Figure 5.18: 10-split partition tree for `Duration` from the summarized epidemic dataset.

After we have identified what interactions are influential, visualization can help us interpret them. In Section 4.8.5 we mention JMP's interactive prediction profiler, and describe how it can be used interactively to find good combinations of factor settings. Another use is for visualizing the interactions themselves: by changing the level for one factor involved in an interaction, you will see a corresponding change in the other, as in Figure 5.19a. This type of behavior can be summarized in a 2-dimensional interaction plot, as in Figure 5.19b. The correspondence between the two follow. The top curve in the right of Figure 5.19a shows the predicted `Duration` as a function of `Transmission Ratio Reduction` while holding `Transmission Ratio` at its lowest value of 0.5. This is the curve labeled as 0.5 in the upper right of the interaction plot in Figure 5.19b. Similarly, the lower right curve in Figure 5.19a shows the predicted `Duration` as a function of `Transmission Ratio Reduction` while holding `Transmission Ratio` at its highest value of 3.5. This is the curve labeled as 3.5 in the upper right of the interaction plot in Figure 5.19b. Its curvature indicates that the metamodel contains a quadratic effect for `Transmission Ratio Reduction`, otherwise this would be a straight line. The curves in the lower left of Figure 5.19b are those associated with setting `Transmission Ratio Reduction` to its lowest and highest levels (0 and 0.75, respectively), and observing the predicted `Duration` as a function of `Transmission Ratio`.



(a) Two snapshots of a prediction profiler.

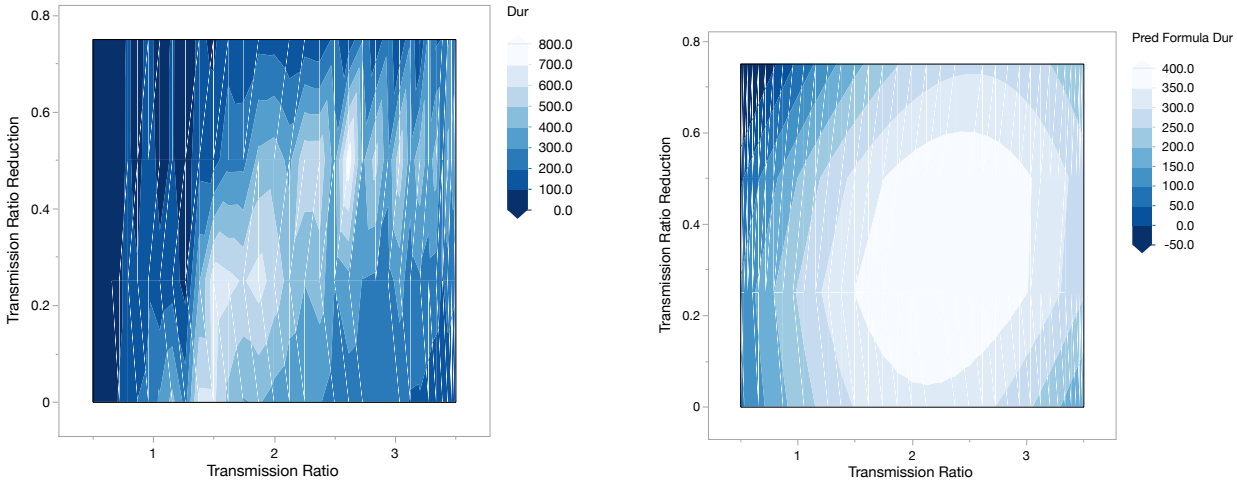
(b) Interaction plot.

Figure 5.19: Visualizing interactions in a metamodel of `Duration` from the epidemic experiment.

We can interpret these interactions as follows. From the upper right of the interaction plot, we notice that the two curves are close together on the right and farther apart on the left. If the disease spreads easily, the epidemic duration rises and then falls as we increase the `Transmission Ratio Reduction`. On the other hand, if the disease spreads less easily, then small interventions have little effect on `Duration`, but `Duration` drops rapidly as the effectiveness of the interventions increases. Looking at the lower left of the interaction plot gives us a different view of the same story. The predicted `Duration` rises and then falls as the `Transmission Ratio` increases. The fact that the curves cross means the answer to "what's the best `Transmission Ratio Reduction`? is "it depends." When `Transmission Ratio` is low, then the epidemic is over sooner if the intervention is highly effective. However, if the `Transmission Ratio` is high, the epidemic will be over slightly faster if we let it run its course without intervention. Of course, as earlier analysis has shown, this comes at the cost of having high proportions of the population become infected. In general, an interaction plot will have one row and one column for every factor that appears in at least one interaction in the metamodel.

Interaction plots are not the only possibilities. We have already used contour plots to indicate how the response surface changes as a function of two factors for the capture-the-flag simulation. Contour plots are also useful for visualizing two-way interactions for more complicated simulations. They can be produced for either the raw or summarized data points. In either case, remember that the raw data (or data summarized by dp) from a k-dimensional factor space is being projected onto a two-dimensional space. Consequently, contour plots give us only partial information and may appear bumpy due to effects of other factors that are masked by projections. The contour plots in Figure 5.20 show how `Duration`

changes as a function of the same two factors for the epidemic experiment when the plots are based on the summarized data or the metamodel predictions, respectively.



(a) Based on summary values for the 148 dps.

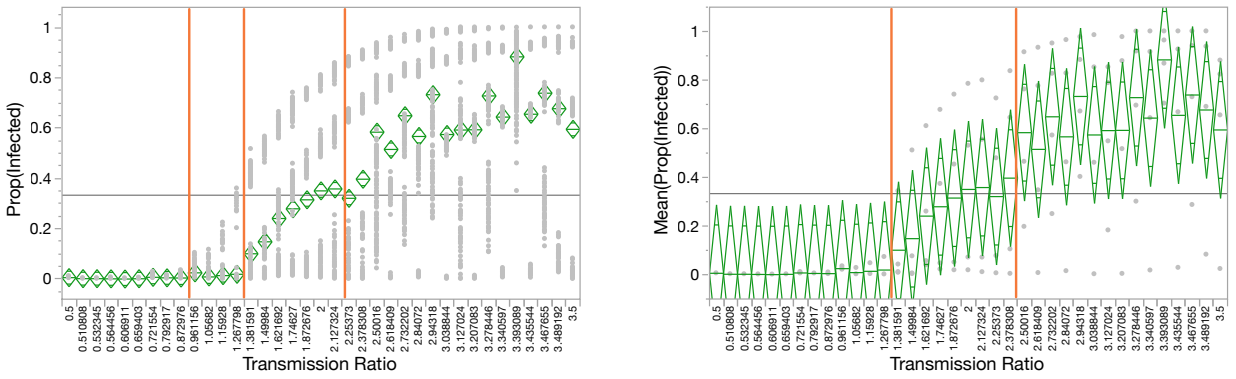
(b) Based on metamodel predictions for the 148 dps.

Figure 5.20: Contour plots of `Duration` versus `Transmission Ratio` and `Transmission Ratio Reduction` from the summarized epidemic dataset. The dark-to-light color scales differ, ranging from 0 to 800 days in (a) and 0 to 400 days in (b).

5.7 Q7: Interesting regions and threshold values?

Inflection points or knees-in-the-curve where the underlying system seems to change its behavior are sometimes referred to as threshold values. Let's return to the epidemic experiment for an example. Figure 5.21 shows two mean-diamond plots, one constructed from the raw dataset and the other from the summary dataset. For any particular transmission ratio (TR) level, the center of the diamond indicates the mean response, and the upper and lower points of the diamond correspond to the upper and lower confidence limits for the mean response. We have added some vertical lines at locations in the curve worth discussing. In Figure 5.21a, there are three lines, separating the transmission ratio levels into four groups. From left to right, the first region shows that TR levels between 0.50 and 0.87 all have response values that are very low—no points are observed above the diamonds. In the second region, corresponding to TR levels between 0.96 and 1.27, the diamonds look essentially the same but we observe some points where the response is substantially higher. There is a clear knee-in-the curve as we enter the third region—the diamonds climb steadily upwards as the TR levels range from 1.38 to 2.12. This knee is interesting. Epidemiology models based on differential equations predate modern computing. Based on those, medical science has known for over a century that TR levels below 1.0 result in the epidemic dying out, while TR levels above 1.0 mean the epidemic will eventually infect the vast majority of the population. However,

interventions that reduce the underlying transmission rate occur for 3/4 of our data, so our knee occurs at an initial TR level greater than 1.0. Finally, in the fourth region a great deal of variability is present in both the diamond locations and in the raw response results. The diamonds appear to bounce around rather than following patterns of little or no change from their neighbors.



(a) Mean-diamond plot for raw dataset.

(b) Mean-diamond plot for summary dataset.

Figure 5.21: Mean-diamond plots from the epidemic experiment. The `Prop(Infected)` (in (a)) or the `Mean(Prop(Infected)` (in (b)) are plotted versus `Transmission Ratio` for the raw and summary datasets, respectively. Vertical lines separate each plot into regions with visually distinct response behaviors.

In Figure 5.21b, we have fewer points due to summarization, so there is less spread in the summary points and the diamonds are much larger. Here we might separate the TR levels into just three regions. The first, for TR levels between 0.50 and 1.27, corresponds to the first two regions in Figure 5.21a. The knee-in-the-curve is still visible as we move to the middle nine TR levels (TR between 1.38 and 2.39)—these show a fairly steady increase in the diamond locations. In the third region, the trend is less pronounced. The diamond locations bounce around, but at high levels of the response. Little, if any, portion of any diamond in this region falls below the overall mean response.

The mean-diamond plot in Figure 5.21a has much of the same information as the side-by-side boxplots in Figure 5.5a. We show them both because each has some advantages. The boxplots do a better job of conveying differences in variability. The mean-diamond plots construct diamonds under the assumption of constant response variance, which is clearly untrue for the epidemic experiment and most other data farming experiments we have seen. However, by focusing on the means, mean-diamonds plots make it easier to visually identify knees-in-the-curve. They are also related to screening techniques that can be beneficial in several contexts—see Section 5.11.2 for more discussion.

Another example of identifying useful knees-in-the-curve comes from a study by Li (2015). Li conducted a data farming experiment on a model with an embedded optimization component to provide a tool and suggested guidelines for Taiwan military disaster relief planners. The planners were interested in improving the efficacy of their relief efforts in the aftermath of

typhoons—events that happened nearly seven times per year over a 57 year period, and are anticipated to increase in frequency and severity as climate changes accelerate. In all, Li varied 33 factors, including the total budget for disaster relief activities and the survival rate (if rescued) for those who were injured and in need of immediate medical assistance once a typhoon passes. The expected number of casualties is the primary measure of effectiveness for the relief efforts.

A graph of `E[casualties]` versus the total budget, for different survival rates, appears in Figure 5.22. Dashed vertical lines have been placed at knees in the curves. The graph shows that the expected casualties decrease sharply as the budget increases from \$0 to \$5 million, and then level off until the budget reaches \$23 million. Further inspection of the detailed simulation output indicates that in these scenarios, additional funds are spent to establish shelters capable of housing displaced people in preparation for typhoons, although this has no effect on reducing the expected number of casualties. Slight improvements are achieved between budgets of \$23 and \$30 million as the additional funds are used to expand pre-positioned emergency health service capacity and commodity supplies. Funding beyond \$30 million yields no further improvement.

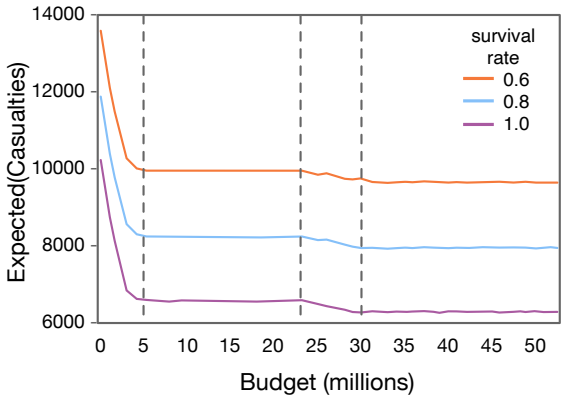


Figure 5.22: Expected(casualties) versus budget for a data farming experiment involving a model of disaster relief efforts, adapted from Li (2015). Dashed vertical lines indicate knees in the curve.

Insights gained from Q7 may be used in several ways:

- They may be worth discussing directly with stakeholders and decision makers;
- They may help focus analysis efforts on interesting regions within existing data; or
- They may suggest modifications to the factor space for subsequent experiments.

We remark that identifying the existence of clear knees-in-the-curve may be more important than determining precisely where they are located, since our simulations are themselves models. However, identifying such a point provides both an opportunity and a caution. In the disaster relief study, the opportunity for budgetary substantial savings is clearly evident—reducing the budget from \$50 million to \$35 million, or from \$20 million to \$10 million, appears to be a no-brainer. The caution occurs as we come closer to the apparent

knee. Thresholds near a steep gradient indicate potentially high risks or rewards, depending on the direction of change. Reducing the budget from \$6 million to \$5 million could be a terrible idea if the true distribution of typhoon severity would place that knee at \$6.5 million. In some contexts, it might be beneficial to conduct some real-world experiments to ensure that the system will operate effectively, or to monitor the results after implementing the recommended solution so the details of the decision can be revisited if necessary.

5.8 Q8: Are any results counterintuitive?

To say something is counterintuitive—in other words, surprising—it helps to first articulate what is intuitively reasonable. This helps mitigate against analysts or stakeholders coming up with explanations or stories to justify results that should, after a closer look, be tracked back to bugs in the simulation model itself or artifacts related to the experiment settings. In Section 1.2.1, we described how asking all stakeholders to jot down a few key expectations, such as "What do you believe the three most important factors will be? How will they affect the response?" More specific guidance on the latter question might be to ask "What direction will lead to overall improvement?" for quantitative factors, or "What levels will be the best?" for qualitative factors. These expectations can help clarify whether or not the results are surprising when it's time for the analysis, as well as provide for lively discussions if different stakeholders have radically different expectations. Ultimately, when a surprising result is found, it should either lead to a bug being fixed (model verification) or intuition being changed (model validation).

Based on decades of experience we wrote the following in Sanchez and Sanchez (2017):

> Large-scale experimentation also helps identify bugs in a model that would otherwise go unnoticed, and may highlight inappropriate assumptions. We used to refer to this as "accidental" verification and validation but now call it "structured" instead.

Here are brief descriptions of some types of surprises we've observed.

- Factors anticipated to be important *a priori* do not show up as important.
- Factors anticipated to be unimportant *a priori* turn out to be very important.
- A factor's main effect is important, but in the opposite direction from what was anticipated.
- Responses exhibit abrupt shifts or behave cyclically as we increase or decrease particular factors.
- Some of the dps appear problematic.
- Surprises arise because some model assumptions are not clear, and different interpretations of what's happening under the hood are possible. This generally reflects a communication breakdown between the model's designers, implementors, and analysts or other stakeholders.

We now provide a few concrete examples where surprises led directly to the identification and subsequent fix of model bugs. In the pilot training study, one factor was the summer cancellation rate due to bad weather. A single value in the original simulation model was intended to account for the prevalence of summer thunderstorms at a particular training location. NPS's SEED Center constructed a data farming wrapper, allowing the cancellation rate and many other embedded model parameters to be explored with data farming experiments. The simulation could then be used more generally as a model of training facilities in other locations, and we varied the summer cancellation rate from 0 to 30% in an experiment involving many other factors. The intuition of everyone involved was that increasing the cancellation rate over this range would dramatically increase the training days required. Surprisingly, it did not show up as an important factor. Delving into this more deeply via the summary dataset, a simple regression of the Mean(training days) versus the summer cancellation rate showed no effect (p-value > 0.68) and the scatter plot showed no evidence that any other type of metamodel would do better. At that point, the model developer delved into the code and found that the line supposed to invoke the summer cancellation rate contained an error. In pseudo code, the original simulation had the cancellation rate logic as:

```
if (winter) then cancellation_rate = winter_cancellation_rate
...
if (!winter) then cancellation_rate = winter_cancellation_rate
```

when it should have been:

```
if (winter) then cancellation_rate = winter_cancellation_rate
...
if (!winter) then cancellation_rate = summer_cancellation_rate
```

This was a simple fix once identified, but it had gone unnoticed for some time.

Another example is from a humanitarian assistance study by Wolf (2003). It involves the movement algorithm in an early version of a time-stepped agent-based modeling platform. The movement took place in a checkerboard fashion. At each time step, every agent would determine their preferred movement direction and move one step (right, left, forward, back, or diagonally) in that direction with some probability. The initial release of the software involved only agents traveling by foot, with a speed ranging from 0 to 100 corresponding to the probability of movement. In a later release, agents were allowed to travel faster (e.g., to include agents traveling in vehicles) and the allowable top speed was increased from 100 to 1000. However, some of the results using gridded designs were surprising. For example, when the top speed of agents in the simulation was increased slightly from 100 to 110, the agents seemed to grind to nearly a halt. After further inspection, it was determined that only the upper bound for the speed in the movement algorithm had been updated, while the lower bound remained at zero. For example, a speed of 101 meant that 1% of the time the agent took two steps, but 99% of the time the agent would take no steps. Intuitively, this should have been a random choice between 1 step and 2 steps, not 0 steps and 2 steps. Figure 5.23 shows the relationship before and after the bug fix.

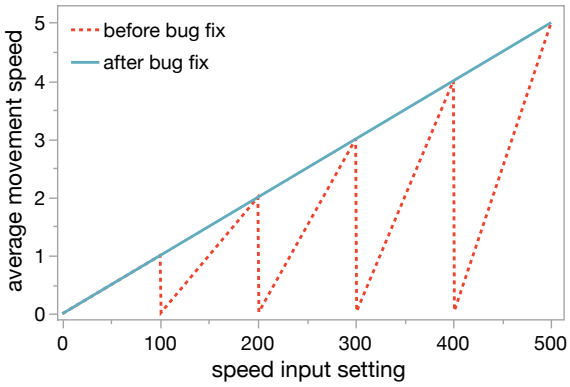


Figure 5.23: Average movement speed in an agent-based modeling platform as a function of the speed input setting, before and after a bug fix.

A third example, also from Wolf (2003), involves a different aspect of movement in an agent-based modeling platform. Here, a movement uncertainty factor is used to inject variability into the movement direction. In other words, although an agent has a preferred direction of movement at a particular point in time, it might choose to move in a different direction. Figure 5.24 plots the average number of civilians fed as a function of their movement uncertainty. The results show a clear outlier when the uncertainty level is zero. Otherwise, low uncertainty leads to higher numbers of civilians fed, and there is a general downwards trend as the movement uncertainty increases. All runs associated with the outlying dp were discarded for subsequent analysis while this bug was investigated.

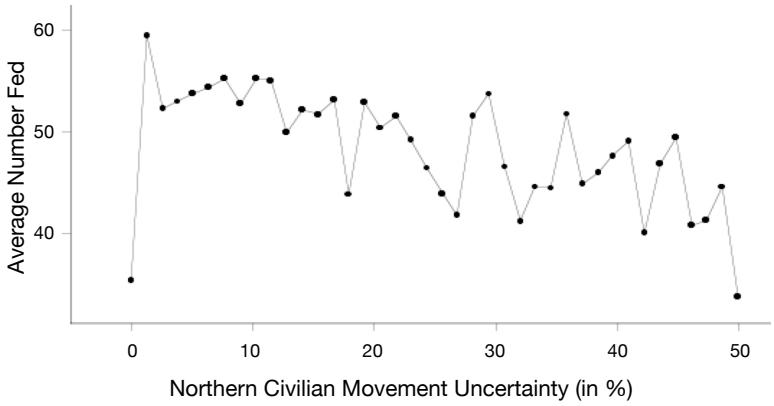


Figure 5.24: Number of civilians fed versus movement uncertainty for these civilians from a simulation of food distribution in humanitarian assistance operations.

These examples show that verification and validation are intrinsic benefits when space-filling designs are used. Bugs such as the ones described above might never have been identified without the systematic exploration these designs provide. In particular, the movement speed problem would have been masked if speeds were only varied within the 0–100 range, or varied as increments of 100. In the third example, if movement uncertainty had been explored at only 2 levels, the conclusion would be that it had no impact.

5.9 Q9: Are there robust configurations or regions?

Although this is listed as question 9 here, we believe it is important enough to have an entire chapter. Chapter 6 delves into the task of identifying robust solutions in depth. For purposes of the discussion in the context of the top 10 list, we stick to a short explanation of the major ideas, coupled with some examples and graphics.

A robust solution is one that consistently yields good performance. In other words, the expected result is "on target" and the variability of the result is low. When using DOE to find robust solutions, we classify the factors into two categories: (i) decision factors that can be controlled by the decision makers, and (ii) noise factors that affect the response(s) but are not controllable by the decision makers. A robust solution is a configuration of the decision factors that works well regardless of the noise factor settings.

Real-world situations abound where robustness is beneficial. For example, computers used to be very sensitive to vibrations and changes in humidity and temperature. Trying to control the operating environment meant building specialized facilities. In contrast, in addition to being much smaller, today's laptop computers are quite robust. They operate inside and outside in a wide variety of microclimates, and can often handle being dropped (although we don't recommend that). Another example of robustness in semiconducting may relate to the number of potential connections on a semiconductor chip. Manufacturers have found that in some cases it is cheaper to produce chips with an extra potential connection that can be used to bypass flaws if needed, than to produce chips with the minimum number of connections and discard those that are not perfect.

Turning to the simulation world, let's first consider the situation where we have a single qualitative factor. Each level of that factor is a distinct alternative—a potential solution to the problem at hand. For example, we might use the `fleet_availability.rb` simulation to model a fleet of delivery drones, but set up an experiment to determine which of two possible vendors should be chosen for drone purchases. Our single decision factor is the vendor, and its settings are simply {A, B}. The vendors could have their own reliability rates and scheduled maintenance cycles. Our primary response is the number of drones available at the beginning of each day. Contractual agreements with major customers mean that we need at least 45 available every morning to provide services. After running the experiment, the results show:

- with Vendor A, 45 are available every day; and
- with Vender B, 40 are available for half of the days, at random, and 50 are available on the other half.

Both vendors have the same expected fleet availability. However, with Vendor A we always meet our contractual obligations, while with Vendor B we do so only half the time. Vendor A is clearly the best—and robust—choice.

More generally, we are likely to have many decision factors and many noise factors. The results may not be so clear cut, and we may have to make trade-offs between the means and the variabilities when comparing alternative solutions. Numeric loss functions can be set up to assess these tradeoffs, as we discuss in Chapter 6.

When we have quantitative factors, we can think of robustness manifesting as large plateaus in the response surfaces. An example from an agent-based simulation of a mobile patrol operating in a remote location appears in Figure 5.25. The patrol carries all of its supplies as it searches for smugglers, and needs to be resupplied by a helicopter on a regular basis to keep the load manageable. The two decision factors plotted are the helicopter speed and the quantity of supplies scheduled for delivery to the patrol in a single helicopter flight. There were numerous other decision factors and noise factors (not shown). A result that was at first surprising was the increase in effectiveness associated with relatively slow helicopter speeds and relatively high days of supply. Tracing it back, the analyst found an interaction effect—at slow speeds, the helicopter's sensors could sometimes detect smugglers and relay their location to the patrol. The patrol, in turn, could successfully act on this information if it carried enough days of supply to head toward that location. A gratifying result is that this improved interdiction capability appeared to be robust, as evidenced by the large, relatively flat region in the upper left of the graph.

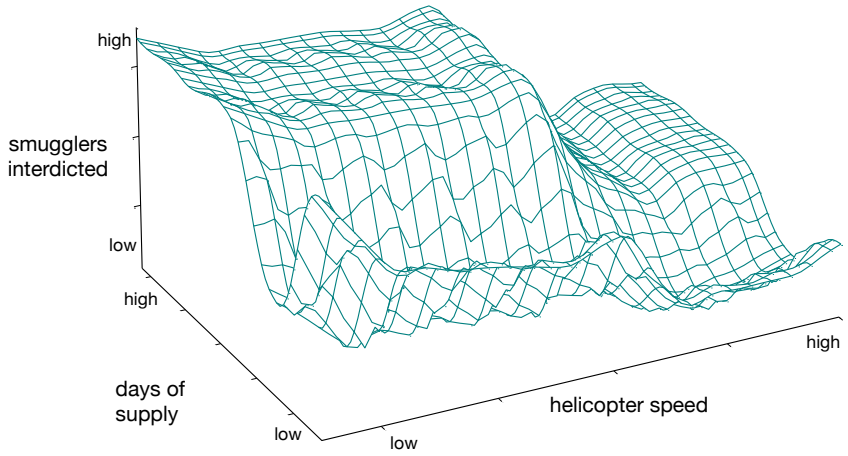


Figure 5.25: Response surface with plateau behaviors for two factors in an agent-based simulation of patrol operations. Steep "cliffs" indicate more fragile operating regions, since small deviations in the factors could produce large changes in the outcome.

Our final example uses data from a model of naval helicopter operations, studied in Marlow et al. (2015) and Marlow et al. (2019). Ideally, each helicopter (referred to as a 'tail') is flown for 10000 hours every year over its 30 year lifetime: 5000 while ashore, and 5000 while embarked. Factors included various maintenance policies, including a heuristic for determining when to rotate helicopters from ship to shore, a maximum variation in the number assigned to each of two squadrons, as well as other factors related to the breakdown, maintenance, and repair processes. Two key annual responses relate to the average and

spread (maximum-minimum) for all tails in the fleet, averaged over the 30 year lifetime of the fleet to obtain the Mean(average) and Mean(spread) in Figure 5.26. Ideally, every tail would accumulate exactly 5000 embarked flight hours every year, so the the Mean(average) would be 5000 and the Mean(spread) would be zero. Figure 5.26 shows this is not achievable. Differences due to the tail rotation heuristic are more noticeable if small variations in the numbers assigned to each squadron are allowed. The third tail rotation heuristic achieves a much lower spread than the other two in all circumstances.

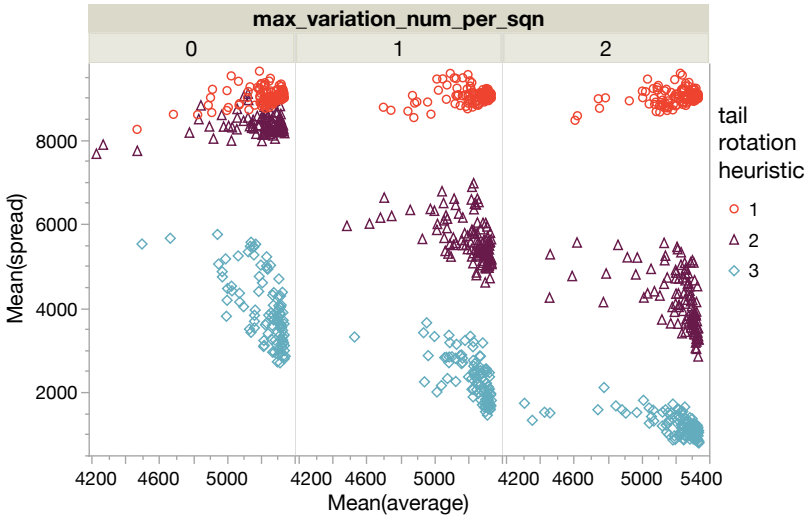


Figure 5.26: Plot of the spread and average annual flight hours for helicopters while embarked.

5.10 Q10: Satisfying multiple objectives

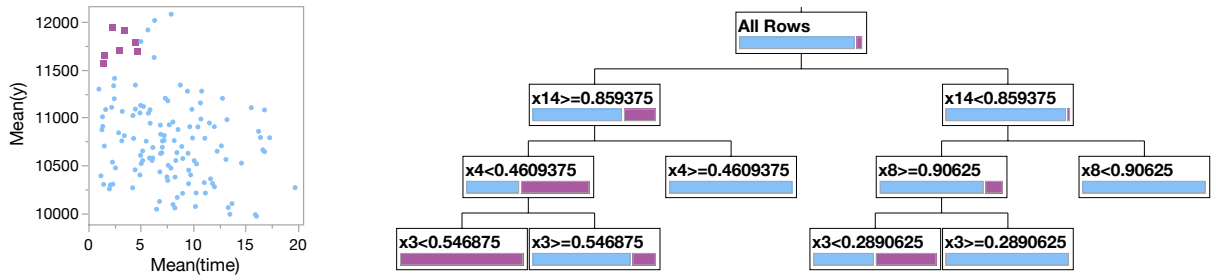
Recall that part of "thinking big" is thinking about more than one response. There are several ways that this can be accomplished. In this section, we briefly describe a few of them using the summary dataset for the big project experiment.

Compare metamodels

If you have constructed metamodels for the key responses, then you can work in the metamodel space. For example, you can create a summary table that shows the impact of factors on different responses to get a better understanding of which factors dominate the decisions. A relative impact table such as that in Figure 4.19 or Figure 5.17 works for partition tree metamodels. More generally, a table that lists factor inclusion in a metamodel, perhaps with indicators of its overall strength and the sign of its main effect, can be constructed and used to help determine whether there are tradeoffs associated with increasing or decreasing the level of a particular factor. See MacCalman et al. (2016) for an example. Graphical techniques that visually display these metamodels can be used to identify regions that are predicted to simultaneously perform well for multiple objectives.

Direct identification

Suppose we are primarily interested in two performance measures, Mean(y) and Mean($time$), and we want to identify solutions that are good for both. One possibility is to set constraints on their values—we could look for dps that satisfy Mean(y) $\geq$ 11500 and Mean(time) $\leq$ 5 and consider these "good" while considering all other dps to be "bad." After creating a new binary response column for good or bad dps, classify each dp as selected or not. You can then construct a new metamodel to find out if you can identify factor conditions that yield good solutions. Figure 5.27 illustrates this approach. The scatter plot in Figure 5.27a shows that there are 7 dps of the 129 total that meet both criteria. The 5-split tree in Figure 5.27b has $R^2 = 0.77$ with a minimum leaf size of 4, and no further splits were possible. It identifies two potential paths to success. The left-most leaf shows that having `x14` ≥ 0.86, `x4` < 0.48, and `x3` < 0.55 yields a leaf comprised entirely of good dps. On the other branch when `x14` < 0.86, the combination of `x8` ≥ 0.91 and `x3` < 0.27 was half successful. Recall from Figure 4.19 that `x4` had a moderate relative impact for `Mean(`y`)` and a very strong relative impact for `Mean(time)`, and very strong relative impacts were also seen for `x14` and `x8` (on `Mean(`y`)`) as well as `x3` (on `Mean(time)`).



(a) Seven dps satisfy both criteria.

(b) 5-split partition tree has $R^2 = 0.769$.

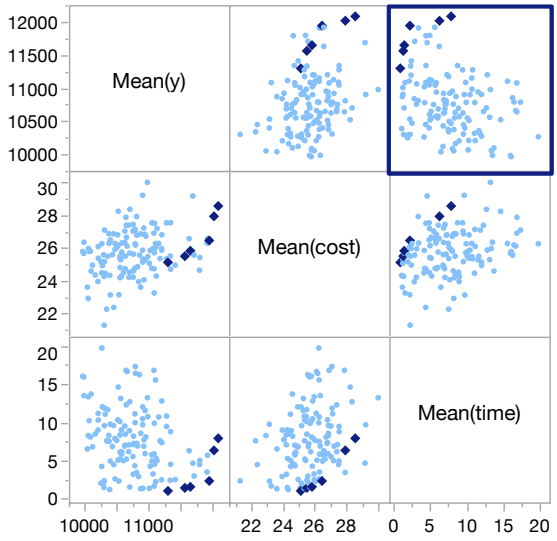
Figure 5.27: Design points that meet criteria of Mean(y) $\geq$ 11500 and Mean(time) $\leq$ 5.

One drawback to this approach is that there may not be any dps that meet all criteria, especially if the number of responses is large. If so, there are a few other heuristics that could be tried. You could relax or remove one or more constraints. You could rank the dps for each of the responses of interest, and then see if any dps are present in all of the top (say) 10 ranks. You could score the dps based on individual metrics and select a set with high combined scores to serve as the "good" subset.

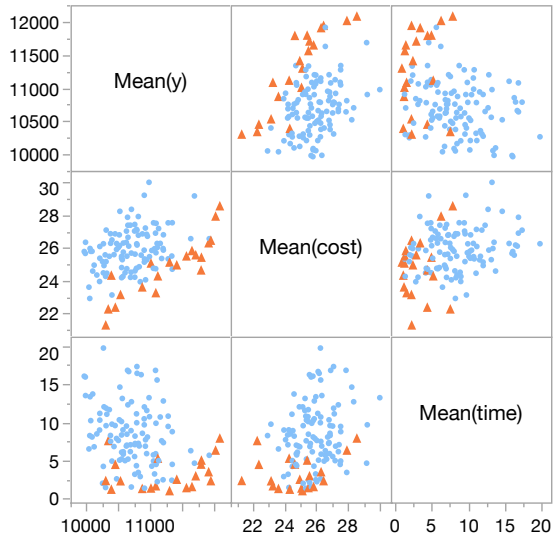
Another drawback is that metamodels may be somewhat fragile if a small number of satisfactory dps are identified. For instance, the lower branches of the tree may be sensitive to the choice of minimum leaf size. In the big project example, the first three levels of the tree (e.g., showing three splits) were identical for leaf sizes between 2 and 7 but there were some differences in the fourth level. Additional experiments might need to be conducted for validation purposes, focused around some of the "good" leaves or branches to confirm that more nuanced findings hold up.

Pareto frontier

Another approach is to identify dps on the Pareto frontier, rather than specify individual constraints on the response values. The Pareto frontier consists of dps that are not dominated by any others. For example, consider the scatterplot matrix in Figure 5.28a. The dark blue diamonds indicate the 6 (of 129) dps on a two-dimensional frontier, as we seek to maximize `Mean(`y`)` and minimize `Mean(time)`. These are considered non-dominated points—any other dp in the data has worse `Mean(`y`)` for the same `Mean(time)`, worse `Mean(time)` for the same `Mean(`y`)`, or is worse in both dimensions. Only 3 of these points are among the 7 highlighted in Figure 5.27a. Deciding which of the Pareto dps is preferable involves tradeoffs. A decision maker primarily concerned with the project completion time would prefer the left-most Pareto point in the upper right corner of Figure 5.28a. A decision maker willing to wait longer can improve on y by moving to other points along the frontier. A scatterplot matrix of points on a three-dimensional frontier of `Mean(`y`)`, `Mean(cost)`, and `Mean(time)` appears in Figure 5.28b. There are 20 points on this frontier, and they are no longer distinctly separate from the dominated points on any of the subgraphs due to projection.



(a) Dark blue diamonds are the 6 points on the Mean(y) and Mean(time) frontier.



(b) Orange triangles are the 20 points on the Mean(y), Mean(cost), and Mean(time) frontier.

Figure 5.28: Scatterplot matrices highlighting the design points on a Pareto frontier for the big project experiment, where high values of y and low values of cost and time are desirable.

One benefit of the Pareto approach is that it presents decision makers with a set of potential solutions that are individually good, but involve tradeoffs. This is often much more palatable than declaring that a single solution is 'best.' The tradeoffs are comparative, so even if decision makers believe that the model may (say) underestimate the time required, they may be comfortable about exploring tradeoffs in terms of weighing the risks versus benefits of accepting a longer completion time while achieving a better final product. A drawback of

the Pareto approach is that the size of the frontier tends to increase rapidly as the number of responses used to define the frontier increases. It may not be helpful to decision makers in these circumstances, unless stakeholders can come to a consensus on a few key responses to consider.

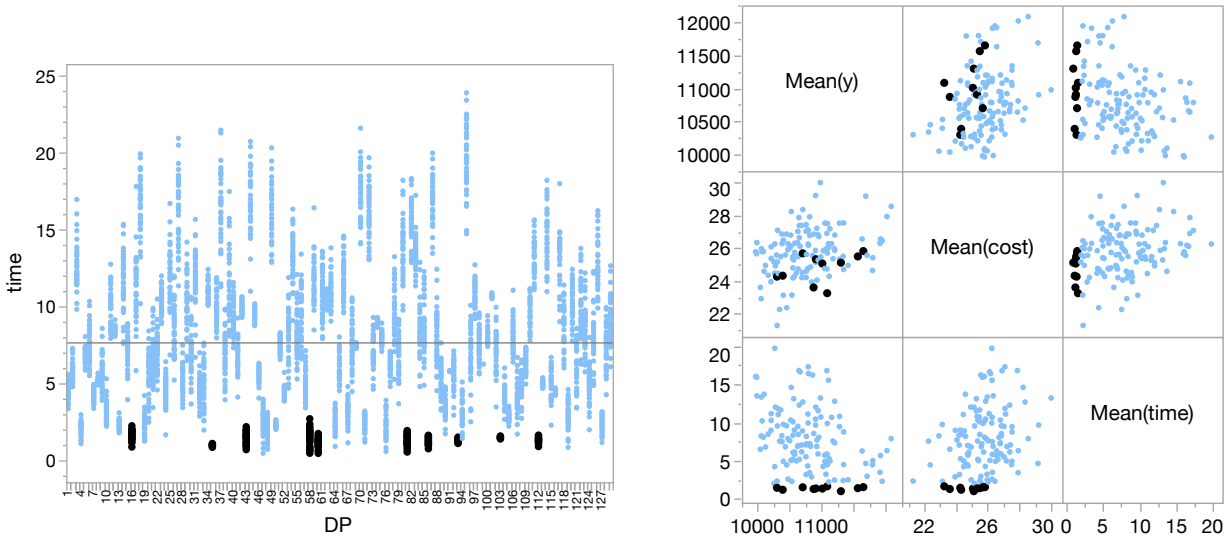
Screening approaches

Another approach for handling multiple performance measures is to first identify a "good" subset of the dps with respect to one response, and use secondary criteria to make the final decision. Screening refers to eliminating a large number of clearly-inferior solutions early. A series of sequential screens works best if the responses can be prioritized so we screen out (and eliminate from further consideration) dps that are inferior with respect to the highest priority response first. We might eventually identify a single solution, or we might stop at some point—perhaps even after the first screening—with a subset of the original dps to be shown to the stakeholders and decision makers.

Multiple comparisons with the best and subset selection are two statistical screening techniques that provide some probabilistic guarantees when applied to a single objective function. These should be implemented with the raw dataset, not the summary dataset. For a more detailed explanation of these procedures, see Section 5.11.2. Here, it suffices to say that either approach will end up selecting a subset of random size that, with high confidence, contains the alternative that is truly best. All alternatives in the selected subset are considered to be good alternatives based on the available data.

We return to the big project example. Instead of using a strict cutoff of 5 for the time limit, we screen the dps to find a subset with good times. The highlighted points in Figure 5.29a correspond to the 10 dps in the selected subset for minimum `Mean(time)`. We highlight these same dps in the summary dataset in Figure 5.29b so you can easily compare them to the graphs from earlier in this discussion.

This approach, like the others, has both advantages and disadvantages. If you stop after a single screening step, the size of the selected subset will tend to be large if there are many alternatives that are not statistically significantly different from the observed best; and tend to be small (perhaps just one) if there are few contenders. This is a nice feature, as it allows decision makers to consider both the other responses from the experiment data as well as qualitative concerns that may not be reflected in the simulation itself, when making a final decision. It could be performed separately for each of the key responses, although there is no guarantee that there will be any overlap in the subsets. One drawback is that the statistical guarantees provided by screening procedures are based on underlying assumptions, such as normally distributed outcomes at each dp, that may not hold for simulation data. It is also not clear what impact an iterative use of screening has on these probabilistic guarantees, particularly when responses are correlated.



(a) All observations in the raw dataset.

(b) Scatterplot matrix for the summary dataset.

Figure 5.29: Black dots correspond to the 10 dps in the subset after screening for minimum Mean(time) in the big project experiment.

5.11 Other data farming considerations

5.11.1 Leverage reproducibility

We described some benefits of reproducibility in Section 3.6.4. We add to that discussion here, in the context of insights that arise from reproducibility.

Columns containing the replication number and random number seed may be useful if the simulation permits user-specified seeds.

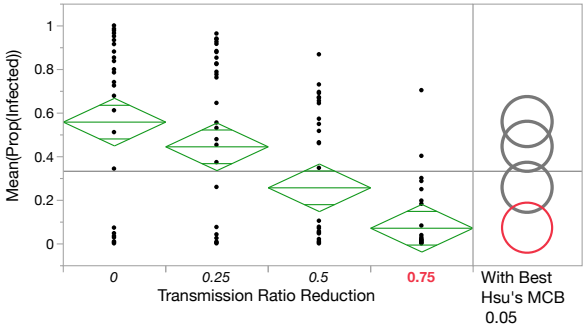
- In the farming for modeling phase, if you have detected a bug from a particular run when using a particular random number seed, you can rerun that case to determine whether the attempted bug fix was successful.
- In the farming for insights phase, you could rerun one or more particular cases if there is a chance to gather more detailed data or run with animation turned on. As discussed in Section 3.6.4, this may lead to new insights.
- The use of animation may also be beneficial in the insights to impact phase. For instance, seeds for producing representative runs could be chosen to reduce the chance that an audience would see an unusual run and incorrectly fixate on some atypical feature as being a key performance driver. Alternatively, playback of both a good and a bad run for a particular design point could help convey the uncertainty or risk associated with that choice. Finally, playback of typical runs from two or more dps (e.g., a good dp and a bad dp) might help convey the impact of making different choices.

5.11.2 Consider screening techniques to identify good alternatives

We provide a bit more detail about two screening approaches introduced earlier in this chapter.

Multiple comparisons with the best

If we temporarily treat a particular factor as qualitative rather than quantitative, a mean-diamond plot can be used to show the differences across factor levels. For such a plot, the center of the diamond is the sample mean, and the ends of the diamond indicate the lower and upper bounds of a confidence interval for that mean. A horizontal line shows the overall mean for the dataset. A mean-diamond plot can be constructed on either the raw dataset or on a (partially) summarized dataset, as long as there are multiple observations for each factor level. As an example, consider the results from the summarized dataset for the epidemic experiment in Figure 5.30a. Keep in mind that the variability in the points reflects changes associated with all factors in the experiment, not just with factor indicated on the x-axis—we are projecting a five-dimensional factor space onto a single dimension.



(a) Mean-diamonds graph.

Comparisons with Max and Min

Level	with Max p-Value	with Min p-Value
0	0.9903	<.0001*
0.25	0.1687	<.0001*
0.5	0.0003*	0.0251*
0.75	<.0001*	0.9996

(b) p-values for comparisons.

Figure 5.30: Mean-diamond plot of `Prop(Infected)` versus `Transmission Ratio Reduction` for the summarized (by dp) dataset of the epidemic experiment. Results of Hsu's multiple comparisons with the best procedure appear graphically in Figure 5.30a and numerically in Figure 5.30b.

The locations of the diamonds in Figure 5.30a show that the average proportion infected decreases as the `Transmission Ratio Reduction` increases. We emphasize that often the visual interpretation suffices. However, it is possible to augment this using a multiple comparison approach, such as Hsu (1981) multiple comparisons with the best (MCB) method summarized visually and numerically in this figure. The MCB method controls the overall α for performing $\ell - 1$ simultaneous tests. The right-most column in the Comparisons table of Figure 5.30b indicates that the mean proportion infected for transmission ratio reductions of 0, 0.25, and 0.5 are significantly different (with p-values < 0.0251) from the best result observed for transmission ratio reduction of 0.75. The comparison circles in Figure 5.30a show this visually. If circles are separated or overlap only slightly, then interactively highlighting

the best highlights all that are statistically indistinguishable from the best. The p-values are not exact because Hsu's MCB procedure assumes the responses have equal variances across the four factor levels. Nonetheless, we see very strong statistical evidence that the largest transmission ratio reduction is superior to the others.

The sizes of the diamonds and comparison circles will always be larger for the summarized dataset than for the raw dataset due to the number of points at each factor level (n_{rep} times larger for the raw dataset). With a large number of replications, the comparison circles may just show up as dots or not appear at all.

Subset selection

An alternative to Hsu's MCB procedure is a subset selection procedure that relaxes the common variance assumption. Many such procedures are possible. One that we have applied in the past is a variant of a two-stage procedure in Nelson et al. (2001) we call NSGS, an acronym based on the authors' last names. NSGS is a two-stage procedure designed to identify the alternative with, for practical purposes, the largest true mean. NSGS is based on a user-specified guarantee P^* on the overall probability of correct selection (PCS) (typically, $P^* = 0.90$ or $P^* = 0.95$) and a user-specified value $\delta \geq 0$ corresponding to the minimum difference between the best and second-best true means that would be worth detecting. If $\delta = 0$ we are restricted to the first stage of the method—a subset selection method, sometimes referred to as a screening procedure. For the remainder of this chapter, we fix $\delta = 0$ and limit ourselves to the first stage. (A two-stage variant of NSGS will be discussed in more detail in Section 6.3.2.)

During the first stage, an equal number of observations is taken from each of the alternatives. Pairwise comparisons are conducted to eliminate from further consideration all alternatives whose sample means are statistically worse than the observed best. This leaves the analyst with a subset of random size from the original alternatives. The subset will tend to be large if there are many good alternatives, and tend to be small if there are only a few good alternatives. The number of elements in the selected subset also tends to be small if the first stage sample size is large.

Now let's apply this subset selection procedure to some data farming results. The procedure could be used for any experiment involving a single qualitative factor, in which case the number of alternatives is simply the number of factor levels. Similarly, since dp is a single qualitative factor, we could use the procedure to identify a subset of dps that are considered "good" in terms of yielding large responses: in such a case, the number of alternatives is the number of dps. After running the first stage of NSGS and obtaining a selected subset S, we will be able to give a very succinct and intuitive summary of the procedure, such as

- With high confidence (PCS ≥ 0.95), the subset S contains the true best dps (if the alternatives are the dps); or
- With high confidence (PCS ≥ 0.95), the subset S contains the true best levels of factor i (if the alternatives are the levels of factor i).

If the subset S contains a single dp (or a single level), then with high confidence we can claim it is the best.

Let's apply subset selection to the summarized dataset used for the MCB procedure in Figure 5.30. Because our design crossed a 4-level factor for `Transmission Ratio Reduction` with a 37-dp FBD for the other four factors, our first-stage sample size is 37. In this case, applying using an overall $\alpha = 0.05$ tells us that the subset of good `Transmission Ratio Reduction` levels is $\{0.75\}$ if low values are good, and the subset is $\{0, 0.25\}$ if high values are good. These agree with the MCB results (in this case) but are not based on an assumption of constant underlying variances.

Chapter 6

Robustness by Design

Key Concepts

- Stochastic systems should not be evaluated solely in terms of mean responses: response variability is also essential to correctly assess the performance and risks of these systems.
- A system is robust if it performs well in a wide variety of circumstances, i.e., we can identify and operate in regions where its performance is consistently good.
- The robust design philosophy distinguishes between decision factors that are controllable in real-world settings, and noise factors that are controllable within the simulation but uncontrollable (or controllable only at great cost) in reality.
- Robustness can be designed into a system by considering the impact of both decision factors and noise factors on its performance and consistency, but policy recommendations and design choices are based solely on decision factors.

6.1 Motivation

What is robust design? It is a system optimization and improvement process that springs from the view that a system should not be evaluated on the basis of mean performance alone. In addition to exhibiting an acceptable mean performance, a "good" system must be relatively insensitive to uncontrollable sources of variation present in the system's environment.

6.1.1 Accuracy vs. Precision

Before diving into a more formal discussion, let's use pictures to illustrate one of the main concepts of robustness. Figure 6.1 shows four different patterns of shots against a target. The two plots on the right-hand side show a shooter with high accuracy: the average shot

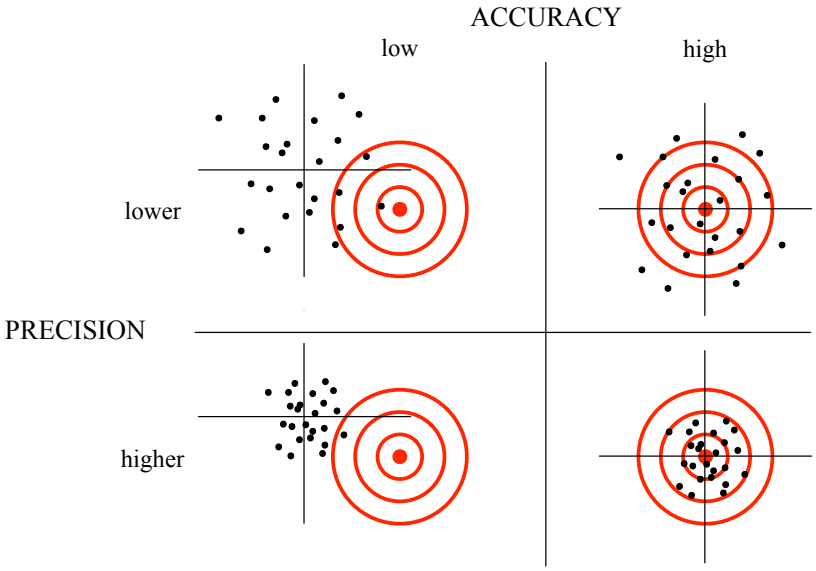


Figure 6.1: Visual representations of accuracy vs. precision

location is at the ideal location—a bulls-eye. Shot patterns for the two plots on the left-hand side are both centered above and to the left of the bulls-eye—in other words, these are less accurate than the plots on the right. Precision describes how tightly the shots are clustered: the two plots on the bottom are more precise than those on the top. If these four graphs represent four alternatives, then we'd clearly prefer the one in the lower right: it has high accuracy and high precision. However, it's less clear which of the other three options is second-best. If the goal is to have a high chance of hitting any of the target area, then the low-precision, high-accuracy alternative in the top right is second, followed by the low-precision low-accuracy alternative in the top left, with the worst alternative being the high-precision, low-accuracy alternative in the lower left. If it is easy to calibrate the aim (that is, to adjust the scope or the sights) then the lower left can quickly be turned into a good option.

There is still room for improvement over any of the options in Figure 6.1. Two such examples appear in Figure 6.2. On the left is an alternative that is substantially more precise than any of the previous four, with very high (though not perfect) accuracy. On the right is an alternative that is perfectly accurate and extremely precise. There may be very little practical difference between these two, so if the improvement in accuracy is expensive it may well not be worth the effort.

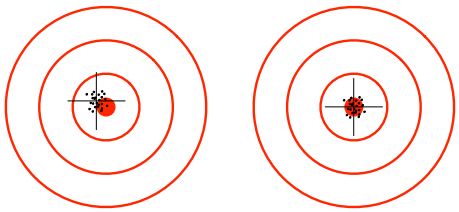


Figure 6.2: Visual representations of two much more robust solutions than the four of Figure 6.1. On the left, a slight lack of accuracy is offset by the improved precision. On the right is the best alternative of the six—perfect accuracy and very high precision, i.e., very low variability.

Some notation and terminology will be necessary before we can engage in a full discussion of the robust design process and its benefits. However, it is important to state up front that the goal of robust design is to lead to better decisions: better in terms of implementation, better in terms of the level and consistency of performance, better in terms of cost, and better in terms of insights into what drives system performance.

6.1.2 Philosophy

The robust design approach originated in quality planning and engineering product design activities (Taguchi and Wu (1980); Taguchi (1987)). Taguchi noted that it was often more costly to control causes of manufacturing variation than to make a process insensitive to these variations, and through the use of simple experimental designs and *loss functions* it was often possible to greatly improve product performance by "building in" the quality. Taguchi's philosophy and strategy were widely praised in both the applied statistics and manufacturing communities (Pignatiello and Ramberg, 1991), but many of the methods and tactics he advocates are often controversial (Box, 1988; Ramberg et al., 1992; Nair et al., 1992). The approach described here and in Sanchez et al. (1996, 1998); Ramberg et al. (1991) combines Taguchi's strategy with response-surface metamodeling techniques. The additional insights that can be gained make this approach particularly beneficial when analyzing simulations of complex systems. In the simulation context, robust design can be viewed from two slightly different perspectives.

One perspective is that simulation is used primarily as a surrogate for a real system due to the cost, time, and risks inherent in making and observing changes in a real system. From this perspective, the total time required to perform an experiment using robust design is greatly reduced, but the designs and analyses used are the same as those that would be applied to a physical system if cost and time permitted. Applications have included product designers' uses of computer models for experimentation in place of physical prototypes, particularly in the semiconductor industry (Sacks et al., 1989; Welch et al., 1990). These experiments have historically involved Monte Carlo simulation, although clearly robustness can be used as a criterion for evaluating discrete-event simulation systems as well. A practical benefit is that if a robust solution is identified in a simulation study, the results are more likely to successfully transfer to the real-world setting.

A broader perspective of the simulation process is also possible. The role of *input analysis*—with a focus on distribution-fitting and parameter estimation—has historically been oriented towards *verification and validation*. We can expand the definition to include inputs that control modeling elements that might previously have been hard-wired into the model structure, such as choices of distributions, protocols/disciplines, or the inclusion/exclusion of sub-model components. This expanded view, combined with experiment design, changes the question of validation from "is the simulation sufficiently valid?" to "what is the impact of changing the model assumptions?"

The first question is difficult because modeling choices are unlikely to be completely accurate and no verification and validation tools can definitively establish model correctness. This is particularly challenging when studying prospective systems, i.e., systems which don't yet exist. In contrast, the second question becomes more manageable. From the expanded perspective, one can view robust design as a process of simulation optimization, where the "best" answer is not overly sensitive to small (and sometimes even large) changes in the system inputs.

Robustness can be used as the basis for ranking a discrete number of alternatives that result from changing the settings of some or all of the input factors. Alternatively, if some or all of the input factors are quantitative, one can construct metamodels of the simulation which describe how the system performance varies as a function of the input factors. There are many approaches to metamodeling, but those described in Chapter 4 work well in the robust design context. Metamodels provide much more information about the underlying system than haphazard investigation of a few alternatives. Thus, if the goal of the analyst is to optimize or improve the model's performance, and flexibility exists in the settings of the decision factor levels (as in prospective studies), then building a metamodel is appropriate.

We are unapologetic advocates for the use of robust design—it can be a game-changer when seeking better decision-making via simulation. Our focus in this chapter is on describing the process for applying it to discrete-event simulation experiments. We begin with a discussion of the terminology used in the robust design approach. We then discuss tactical issues, such as appropriate experimental designs, metamodel construction, analysis options, and robust design identification.

6.2 Notation and terminology

In this section, we present notation and discuss the terminology related to robust design.

6.2.1 Decision factors and noise factors

In systems where stochastic variation is present, the response exhibits random fluctuation or variation. In order to achieve systems or products for which the variation around some target

value is low, several steps are necessary. First, one must identify factors in the system that are anticipated to affect the system response. Factors in real-world systems are classified as *decision factors* or *noise factors*. In simulations there can be model components that don't exist in the real-world, which we refer to as *artificial factors*. We will continue using the notation $X_1, \ldots, X_k$ to represent the k decision factors of interest. We denote w noise factors as $W_1, \ldots, W_w$, and a artificial factors as $A_1, \ldots, A_a$.

Decision factors are those which are (or will be) controllable in the real-world setting being modeled by the simulation. Noise factors are not easily controllable, or are controllable only at great expense in the real-world setting. Noise factors include sources of variation within the real-world system (i.e., within a manufacturing plant) as well as exogenous factors (such as customer and supplier characteristics). Finally, artificial factors are simulation-specific characteristics such as the initial state of the system, the warm-up period (truncation point), termination conditions (run duration), and random number stream(s) (seed values, whether common or antithetic pseudo-random numbers are used, etc.) which have no real-world analogue. As a more concrete example, in a simulation of search-and-rescue operations after a natural disaster, the decision factors might include choices of the communication systems, available equipment, or number of people on the rescue team. Noise factors might include weather conditions, the number and location of those in need of rescue, and the skill levels of the currently available emergency medical technicians. An alternative to an exploratory analysis that seeks to understand how these noise factors affect the responses is a *robust design* approach, where the goal of the experiment(s) is to identify system configurations of the decision factors that yield good performance across the plausible range of noise factor settings—in other words, to identify *robust* systems, rather than systems that are effective only in specific circumstances.

The distinction between decision factors and noise factors is often recognized in simulation experiments, and can be leveraged to advantage when studying the system. The classification is important in several ways. It is necessary for determining system robustness, and also presents an opportunity for reducing the number of runs required by concentrating sampling efforts on assessing decision factor effects, since these are the only ones we can control in the real-world system. The additional layer of control made possible by the artificial factors can also be exploited in the experimental design (Schruben et al., 1992). This is not new—it is the basis of many variance reallocation techniques. However, manipulating artificial factors is not required. An important consideration for the simulation community is that the robust design philosophy explicitly requires analysts to consider variances, as well as means, in assessing system performance.

The classification of factors as either decision or noise factors may affect the choice of the design. Generally, we are interested in fitting metamodels that explain the relationship between the decision factors (and their interactions, etc.) and the response. Interactions among noise factors may affect the variability of the response but are not of direct interest, while (noise factor)×(decision factor) interactions will manifest as unequal response variances across different decision-factor combinations.

The initial characterization of a factor as decision or noise might change over time. For example, consumer demand for items might be considered uncontrollable in the short term, but we know that marketing efforts can affect it in the long run. In search-and-rescue operations with a fixed number of rescuers, the addition of a new technology may end up reducing the noise variability or even turning it to a decision variable. For example, suppose that cockroach-sized sensor drones are able to enter collapsed buildings to seek signs of life and direct rescue workers to them. In this case, the number of drones that augments the human team becomes a decision factor. While there is still noise associated with the time to complete the search, one source of that noise has been partially controlled.

6.2.2 Responses, targets, and goals

The analyst begins by specifying some performance characteristic of special interest, and an associated target value designated by the Greek letter τ (tau). Common measures are related to system throughput or system states, such as the waiting time or number of customers in queueing systems, although cost could also be used as a performance measure. In general, the pattern of the performance characteristic's fluctuation around the target value will differ across these configurations. The cost of this fluctuation must be measured in order to optimize or improve the system. Since end-users will incur costs if system performance deviates from the target, the evaluation criterion is sometimes referred to as the *loss to society* (a philosophical perspective) or the *long term business loss* (when for-profit companies focused on the bottom line are worried about the cumulative effect of disgruntled customers).

We assess responses based on one of three types of goals relative to the target τ:

- the bigger the better;
- the smaller the better; or
- the closer to τ the better.

An example of the first might be to plan a portfolio mixture to make our retirement portfolio as large as possible—achieving more than our stated goal is just fine with us, but we are unhappy with outcomes below our target, and progressively more unhappy based on how big the shortfall is. An example of the second might be to plan warehousing, inventory, and distribution chains to streamline the logistics delivery process by reducing delays. An example of the third might be to use machining or additive manufacturing to produce a part with a specified nominal length.

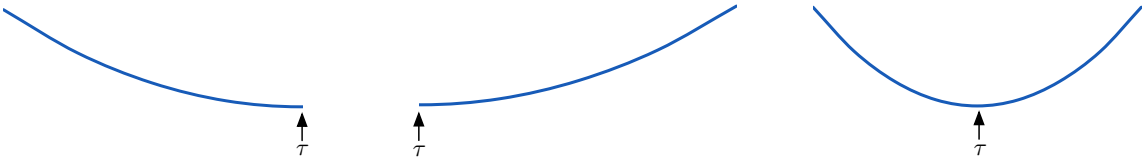
An ideal configuration would result in the performance characteristic's mean equal to τ and its variance equal to zero. This is not usually achievable, so we seek a numerical method for trading-off performance mean and variability. In other words, we need a numerical measure that finds a good balance between accuracy and precision.

6.2.3 Loss functions

In this book we will usually use a quadratic loss function, which (in many cases) is a reasonable surrogate for the "true" underlying loss function which may be difficult or impossible to specify exactly. A quadratic loss function has the nice property that penalties for outcomes near τ are quite small, but progressively larger penalties are incurred as the magnitude of the distance from τ increases.

Quadratic loss function: conceptually

Consider Figure 6.3. The loss function in Figure 6.3a reflects the goal of "the bigger the better" (in terms of y), where an upper bound on y is used for τ. An example would be letting y denote the number of vehicles available at the beginning of a day for the fleet availability model `fleet_availabilitiy.rb`. Clearly, the entire fleet constitutes the maximum possible value for y, and is ideally where we would like to see the system performance. Figure 6.3b reflects the goal of "the smaller the better" (in terms of y), where τ is a lower bound on y. An example would be letting y denote the number of new infections in the epidemic model `epidemic.rb`. We know that y is a nonnegative integer, and ideally would like it to be zero. Figure 6.3c reflects the goal of "the closer to τ the better" when the y both above and below the target value are possible. An example would be letting y denote the distance traveled by gummy bear in the `gummies.rb` simulation if we are attempting to achieve a specific launch distance τ.



(a) One-sided loss, high values are desirable, τ is the maximum possible value of y.

(b) One-sided loss, low values are desirable, τ is the smallest possible value of y.

(c) Quadratic loss function.

Figure 6.3: Variants of quadratic loss functions.

Quadratic loss function: mathematically

Quadratic loss functions have some nice mathematical properties. Let $\boldsymbol{x}$ and $Y_{\boldsymbol{x}}$ denote a vector of decision factor settings and the associated performance characteristic, respectively, and let $\boldsymbol{\Omega}$ denote the noise factor space. If we assume that no loss is incurred when $Y_{\boldsymbol{x}}$ achieves the ideal state (τ), a quadratic loss function $\mathcal{L}$ has the form:

$$\mathcal{L}_{\boldsymbol{x}} = c(Y_{\boldsymbol{x}} - \tau)^2 \tag{6.1}$$

fora specified constant c and target τ. The expected loss (over the noise factors) is then

$$E_{\Omega}[\mathcal{L}_{\boldsymbol{x}}] = cE_{\Omega}\left[(Y_{\boldsymbol{x}} - \tau)^2\right].$$

For the remainder of this book, we will suppress the Ω subscript and simply write this as

$$\begin{aligned}
E[\mathcal{L}_{\boldsymbol{x}}] &= cE\left[(Y_{\boldsymbol{x}} - \tau)^2\right] \\
&= cE\left[((Y_{\boldsymbol{x}} - E[Y_{\boldsymbol{x}}]) + (E[Y_{\boldsymbol{x}}] - \tau))^2\right] \\
&= cE\left[(Y_{\boldsymbol{x}} - E[Y_{\boldsymbol{x}}])^2 + 2(Y_{\boldsymbol{x}} - E[Y_{\boldsymbol{x}}])(E[Y_{\boldsymbol{x}}] - \tau) + (E[Y_{\boldsymbol{x}}] - \tau)^2\right] \\
&= c\left(E\left[(Y_{\boldsymbol{x}} - E[Y_{\boldsymbol{x}}])^2\right] + 2E\left[(Y_{\boldsymbol{x}} - E[Y_{\boldsymbol{x}}])(E[Y_{\boldsymbol{x}}] - \tau)\right] + E\left[(E[Y_{\boldsymbol{x}}] - \tau)^2\right]\right).
\end{aligned}$$

Note that:

- both adding and subtracting $E[Y_{\boldsymbol{x}}]$ inside the quadratic term does not change the equality;
- the first expectation of the final line is the definition of $Var(Y_{\boldsymbol{x}}$;
- $E[Y_{\boldsymbol{x}}]$ and τ are both constants, and $E\left[Y_{\boldsymbol{x}} - E[Y_{\boldsymbol{x}}]\right] = 0$, so the second expectation evaporates; and
- since $E[Y_{\boldsymbol{x}}]$ and τ are both constants, the third expectation becomes $(E[Y_{\boldsymbol{x}}] - \tau)^2$.

The net result of these observations is that expected quadratic loss can be decomposed into the two components:

$$E[\mathcal{L}_{\boldsymbol{x}}] = c\left(Var(Y_{\boldsymbol{x}}) + (E[Y_{\boldsymbol{x}}] - \tau)^2\right). \tag{6.2}$$

This shows that expected loss is proportional to the sum of two components: the variance of $Y_{\boldsymbol{x}}$, and the squared deviation between the expected performance $E[Y_{\boldsymbol{x}}]$ and the target τ. (Note the similarity to mean squared error.) If the expected performance is on-target and has no variability, the loss is zero. Increases in either component contribute directly to increased loss.

Of course, we do not know the true characteristics of the random variables $Y_{\boldsymbol{x}}$ and $Loss_{\boldsymbol{x}}$, but we can estimate them by observing or simulating m values of the response, $y_{\boldsymbol{x},1}, \ldots, y_{\boldsymbol{x},m}$. (For physical experiments, we would collect m physical observations at the design point $\boldsymbol{x}$.) The observed loss l for any particular $y_{\boldsymbol{x}}$ value is

$$l_{\boldsymbol{x}} = c\left(y_{\boldsymbol{x}} - \tau\right)^2. \tag{6.3}$$

When we have access to all individual y values, there are two methods for estimating the expected loss.

Method 1: Calculate $l_{\boldsymbol{x},i}$ for each value of $y_{\boldsymbol{x},i}$, and compute the average $\bar{l}_{\boldsymbol{x}}$ as follows:

$$\bar{l}_{\boldsymbol{x}} = \frac{1}{m}\sum_{i=1}^{m} l_{\boldsymbol{x},i}. \tag{6.4}$$

The result is an unbiased estimate of the expected loss.

Method 2: Calculate the sample average and sample variance of the $y_{\boldsymbol{x},i}$ values as follows:

$$\overline{y}_{\boldsymbol{x}} = \frac{1}{m}\sum_{i=1}^{m} y_{\boldsymbol{x},i} \qquad \text{and} \qquad s^2_{\boldsymbol{x}} = \frac{1}{m-1}\sum_{i=1}^{m} \left(y_{\boldsymbol{x},i} - \overline{y}_{\boldsymbol{x}}\right)^2 .$$

An estimate of the expected loss is then derived by plugging these estimates of the mean and variance into equation (6.2):

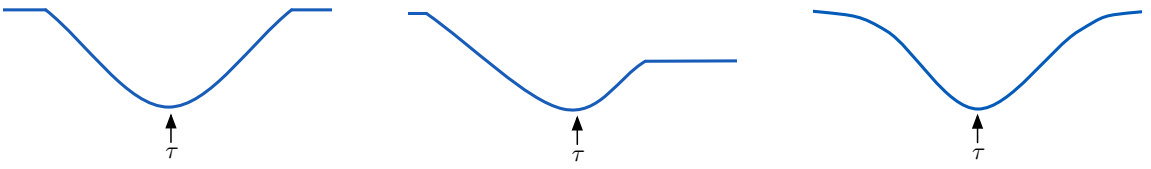
$$\widehat{\mathcal{L}}_{\boldsymbol{x}} = c\left(s^2_{\boldsymbol{x}} + \left(\overline{y}_{\boldsymbol{x}} - \tau\right)^2\right). \tag{6.5}$$

The estimated losses for the two methods differ slightly. They would be algebraically identical if we used the maximum likelihood estimate $\hat{\sigma}^2$ in place of the unbiased estimate s^2 in equation 6.5. The difference between the variance estimates is minimal, particularly if m is large. We assert that ease and consistency are more important than the particular formula chosen, and consider the use of either $\hat{\sigma}^2$ or s^2 equally valid and useful for practical purposes.

Other loss functions: conceptually

Loss functions other than quadratic can be used when appropriate, but won't have the same decomposition into variance and squared deviation from target. The possibilities are endless. Taguchi purportedly used over 60 loss functions, many of them proprietary, when he worked at Toyota.

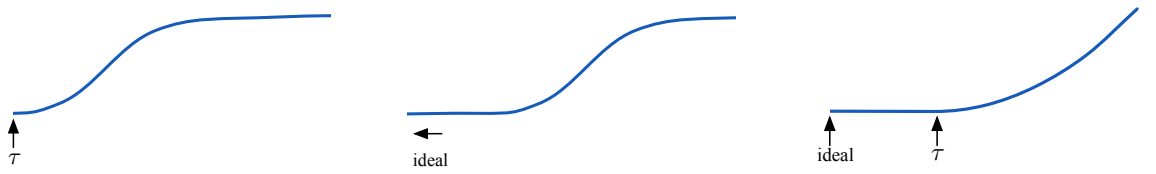
A few two-sided alternatives are shown in Figure 6.4. Unlike the quadratic loss function (for which the loss is unbounded), all three of these cap the maximum loss which may be more realistic in practice. Figure 6.4a is a quadratic function near τ but truncated once the loss reaches its capped value. This might be an appropriate loss function for manufactured parts that will be discarded if their length is outside symmetric specification, where the loss would be bounded by the cost of raw materials, labor, and opportunity costs of lost time and material. The loss function in Figure 6.4b is similar, except that it is less costly to make the part too big than to make it too small. This might be appropriate if an overly-large part could be reworked at a fixed cost. Figure 6.4c is an inverted Normal distribution. Using a smooth curve avoids abrupt transitions associated with reaching the bound in (a) or (b).



(a) Truncated quadratic loss. **(b)** Asymmetrical loss function. **(c)** Inverted Normal loss.

Figure 6.4: Examples of some other two-sided loss functions.

Examples of some other one-sided loss functions appear in Figure 6.5. In Figure 6.5a we have half of an S-shaped curve, such as a logistic function. Note that τ is the smallest possible



(a) S-shaped loss function with a hard lower bound on y and an asymptotic upper bound on loss.

(b) One-sided, monotonically increasing loss function where low values are desirable.

(c) One-sided loss, low values are desirable, τ is greater than the ideal (but unachievable) value.

Figure 6.5: Examples of some other one-sided loss functions.

value for y, so the goal of "the smaller the better" is the same as the goal of "the closer to τ the better." Figure 6.5b shows an S-shaped loss function that extends to $\pm\infty$ over x but is asymptotically bounded above and below for y; this could be used when low values are desirable. Real-world losses associated with several specific values of y (such as those associated with 25%, 50%, and 75% of the asymptotic maximum loss) would need to be determined to write the equation for such a curve. Figure 6.5c is flat (with $loss = 0$) for y values between the ideal and the target τ. This might be used if the ideal value is not considered achievable on average, but τ is a suitably aspirational goal. For example, in the epidemic simulation `epidemic.rb` the ideal number of new infections is zero, but it may be better to target some number τ based on a small percentage of the population of interest. However, replications where the epidemic does fizzle out with fewer than τ new infections should not incur any loss.

Other loss functions: mathematically

As long as we have a well-defined function that maps y to its loss value, we can use Method 1 of equation 6.4 to estimate the average loss. For example, consider the following one-sided loss associated with the result $y_{\mathbf{x},i}$ obtained in the ith replication of design point $\mathbf{x}$:

$$loss_{\mathbf{x}} = \begin{cases} 0 & \text{if } y_{\mathbf{x},i} < 10 \\ 3(y_{\mathbf{x},i} - 10)^2 & \text{if } 10 \leq y_{\mathbf{x},i} < 50 \\ 4800 & \text{if } y_{\mathbf{x},i} \geq 50 \end{cases}$$

We use this formula to calculate the individual $y_{\mathbf{x},i}$ for each replication. Then, as before, we simply take the average of the $y_{\mathbf{x},i}$ for the replications at design point $\mathbf{x}$ to calculate the average loss from our simulated values. This is an unbiased estimate of the expected loss. However, we cannot decompose this into separate components for the mean and variance (or mean and standard deviation) if the loss function is not quadratic over its entire support.

6.3 Qualitative factors

In Chapter 3 we focused on quantitative factors—either continuous-valued or discrete-valued. We brought up the idea of qualitative factors in Chapter 4. Designs for a single qualitative factor are quite simple: each level corresponds to a potential alternative. For example, we might be considering a limited set of vendors from which to purchase a product, a limited set of policy alternatives for dealing with an epidemic outbreak, protocols for deciding which customers have priority in a service system, or alternatives for preventive maintenance scheduling. If we have a small number of qualitative factors k, with $\ell_1, \ldots, \ell_k$ levels, respectively, we could cross the levels for each and treat the resulting $\ell_1 \times \ell_2 \times \ldots \times \ell_k$ combinations as the levels associated with a single qualitative factor.

6.3.1 Visual insights for stylized example with a single factor

Consider what robust design looks like with a single *qualitative* factor. Perhaps we are limited to four approved vendors for supplying a particular part. This corresponds to a single qualitative factor (vendor) with four levels (A, B, C, and D), and if there is no variability in the product for any individual vendor we might observe the results in Figure 6.6. Several targets are shown, because the ideal value may depend on the application. Note that if the target is τ_1, then vendor A achieves this exactly and vendor B a very close competitor, followed by C and then D. If the target is τ_2, then C is the low-loss choice, followed by D, B, and then A. If the target is τ_3, then A is the low-loss solution but there is room for improvement. Perhaps target τ_3 was chosen with the goal of making y as small as possible. Once again,B is a close runner-up while C and D are again inferior options. If the target is τ_4, then the alternatives from best to worst are D, C, B, and A.

Figure 6.6: Responses (ys) are shown for four different levels of a single qualitative factor for deterministic systems, representing four different vendors, with no noise.

Now let's consider the effect noise has on a qualitative scenario. Figures 6.7 and For illustration purposes these are (truncated) normal distributions, but in general they could have any shape. In Figure 6.7, the ranking of the alternatives is unchanged from the deterministic case—we can see from Equation 6.2 that if the variance component of quadratic loss is the same for all cases, low loss will be determined solely by distance from the target. If the target is τ_1 then A is best, followed by B, C, and D. Similarly, if the target is τ_4 then D is

preferred, followed by C, then by B and A which are similar but far inferior options. 6.8 superimpose probability distributions on the outcomes for each of the four vendors.

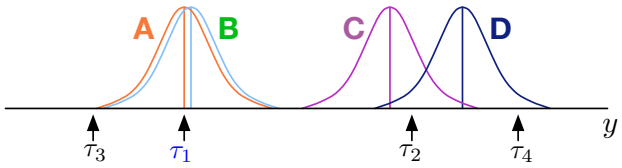


Figure 6.7: Responses (ys) are shown for four different levels of a single qualitative factor for stochastic systems. The noise variability is the same for each vendor.

The rankings can change if if the variability is not constant across the alternatives. In Figure 6.8, B is the most robust choice when the target is τ_1: it has much less variability, and is more likely to be close to τ_1 than any other option. If the target is τ_4 then the low-loss solution is D—even though it rarely yields results near τ.

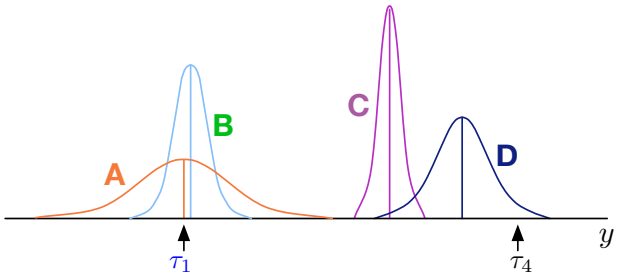


Figure 6.8: Responses (ys) are shown for four different levels of a single qualitative factor for stochastic systems. The noise variability varies from vendor to vendor.

Of course, the outcomes from simulating a particular alternative need not be normally distributed, and the outcomes from different alternatives may follow different distributions. Figure 6.9 illustrates one such situation.

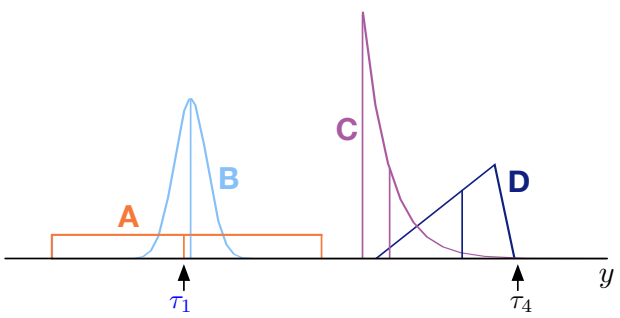


Figure 6.9: Responses (ys) are shown for four different levels of a single qualitative factor for stochastic systems, representing four different vendors. The noise variability and the noise distribution varies from vendor to vendor.

Once we calculate the losses, we need to convert the results to a recommendation with some type of statistical assurance. In the metamodeling context, we can use stepwise regression involving our single qualitative factor to determine which of the alternatives (factor levels) differ significantly from the others. Alternatively, we could use partition tree metamodeling to split the alternatives into different leaves, where the minimum size of a leaf can be chosen to control the minimum number of alternatives in any single leaf. A third method is that of multiple comparisons with the best, which can yield a single alternative if one has much lower loss than all others, or return many alternatives if several have similar loss performance. Finally, we can use a selection procedure intended to compare systems in terms of their losses. See Nelson (2022) or Sanchez and Sanchez (2020) for more details.

6.3.2 Numerical details for single-factor, stylized example

simulate it yourself

`vendor.rb`
: simulate random outcomes y_i from several vendors, where the number of vendors and each vendor has its own user-specified distribution's shape and parameters

`batch_runner.rb`
: a datafarming gem with interactive prompts for running multiple replications of a simulation

Figures 6.6 through 6.9 do not have scales on the y axis. Parameters for the underlying distributions appear in Table 6.1. Since these are known, we can calculate the 'true' E[loss], as shown in the final row of the table.

Table 6.1: Ground truth for the vendor selection problem of Figure 6.9.

Vendor	A	B	C	D
distribution shape	Uniform	Normal	Shifted Exponential	Triangle
μ	10.0	10.1018	13.1055	14.2
σ	1.2124	0.2461	0.4	0.4637
E[loss $\mid \tau = 10.0$]	1.470	0.071	9.804	17.855
E[loss $\mid \tau = 15.0$]	26.470	24.053	3.749	0.855

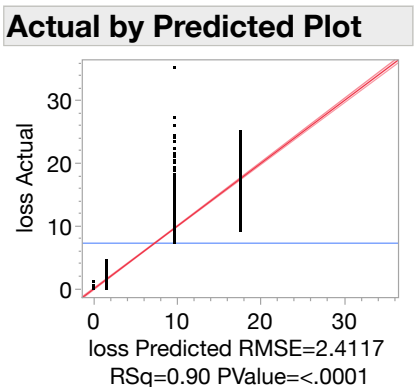
The numerical results for E[loss] agree with the earlier discussion: Vendor B is preferred if the target is $\tau_1 = 10$, and Vendor D is preferred if the target is $\tau_4 = 15.0$. Note that if we had just considered the means when looking at $\tau = 10$, we would have chosen Vendor A and

incurred an actual loss 20 times larger than we would have incurred by choosing Vendor B! This is a far cry from thinking of B as slightly worse than A in terms of the mean E[Y], and perhaps implicitly assuming that the same would hold for the E[loss] values.

Although simulation is not needed when we know the ground truth, it is still useful for illustration purposes. We randomly simulated 1000 observations from each of the vendors. You can generate your own sample using the `vendor.rb` simulation for hands-on experience following along with the analysis. We describe three ways of analyzing the results that provide some statistical assurance of the findings: regression metamodeling, partition tree metamodeling, and a multiple comparisons approach that includes a nice graphical display. Because our only decision factor (the preferred vendor) is qualitative, there is no benefit to fitting separate metamodels to the response mean and standard deviation. Consequently, after reading the data into our favorite statistics software, we can add two new columns: one for the loss associated with each of the two potential target values τ_1 and τ_4. We focus on the former, and leave the latter for the reader wishing additional practice.

Regression metamodeling

We can fit a metamodel of $loss_y$ (when $\tau = 10$) using `vendor_id` as the single (qualitative) decision factor. Partial output appears in Figure 6.10. From this, we see that the overall metamodel is statistically significant ($p - val < 0.0001$) and differences among the vendors account for 89% of the variability in the y values. The plot of actual vs. predicted values in Figure 6.10a shows four distinct vertical stripes, one associated with each vendor. The estimated coefficients and tornado plot in Figure 6.10b show there are significant effects due to different vendors. Recall, however, that for a qualitative factor, the level effects can be presented in different ways, e.g., as differences from a baseline category or as differences from the overall average loss in the dataset.



(a) Plot of actual vs. predicted values.

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	3	200341.84	66780.6	11481.46
Error	3996	23242.28	5.8	Prob > F
C. Total	3999	223584.12		<.0001*

Sorted Parameter Estimates

Term	Estimate	Std Error	t Ratio		Prob>|t|
vendor_id[B]	-7.197342	0.066048	-109.0		<.0001*
vendor_id[A]	-5.745101	0.066048	-86.98		<.0001*
vendor_id[C]	2.4849637	0.066048	37.62		<.0001*

(b) Metamodel details.

Figure 6.10: Metamodel of average loss for four vendors using $\tau = 10$.

The differences between the vendors are easier to visualize graphically. Snapshots of the profiler for each vendor appear in Figure 6.11, along with confidence intervals for the mean losses for each vendor. The 95% confidence intervals for mean losses do not overlap, indicating statistical significance.

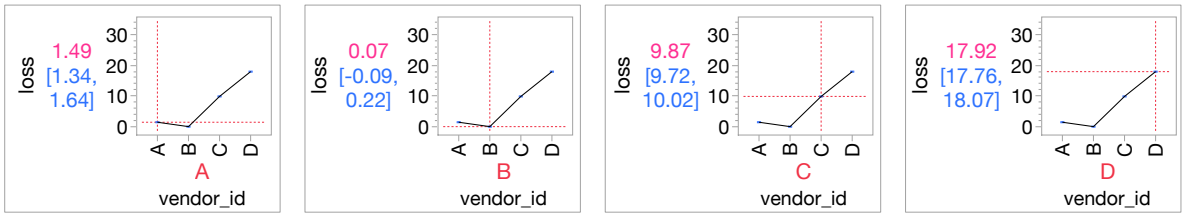


Figure 6.11: Graphical depiction of the differences in losses for four vendors.

Statistical significance is necessary but not sufficient for ensuring that the results have practical significance. In this example, the loss of the worst vendor (vendor D) is over 250 times as large as that of the best. Clearly, the differences among vendors are of practical importance in this example. The loss associated with the second-best vendor (vendor A) is over 20 times as large as that of vendor B, though graphically they appear much more similar to each other than to vendors C and D.

We remark that stepwise regression is a viable approach. If, say, some vendors are not different from each other, those levels will be combined into a single term. For instance, if we added Vendor E to the study and reran the analysis, but Vendor E yielded y values according to a distribution (nearly) identical to Vendor C, then stepwise regression might lump C and E together as a single effect. Alternatively, if Vendor E's y values were distributed nearly identically to those of Vendor B's, then the simplified stepwise model would make it clear that we have two good choices from a practical perspective.

Partition tree metamodeling

A partition tree metamodel for the vendor selection problem appears in Figure 6.12. The results are the same as those for using the regression metamodeling approach, but the partition tree may be preferable for communication purposes.

Note that the mean losses for the four leaves are identical to the predicted values in Figure 6.11. The column contributions (not shown) are not interesting when a single factor is involved, because all splits occur on the same column. We remark that pruning this tree removes the split on the lower left, and keeps B and A in the same leaf while reducing R^2 from 0.893 to 0.888. This indicates that from a practical perspective, we may want to alert management that either Vendor A or Vendor B may be a viable source of supply. Regardless, the vendors are not identical in terms of overall loss. The vendor factor has a "very very strong" relative impact according to the categorization in Section 4.8.5.

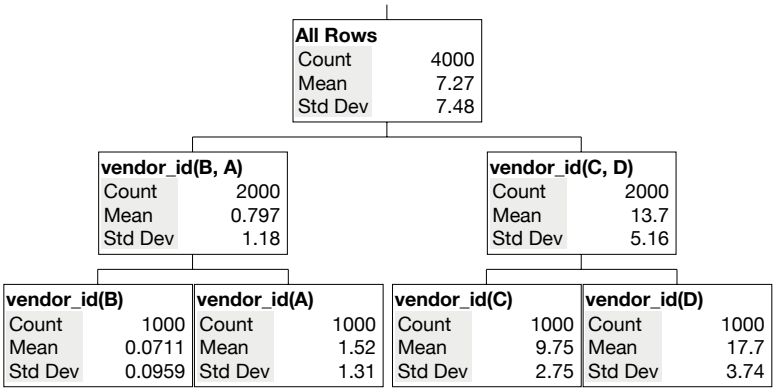
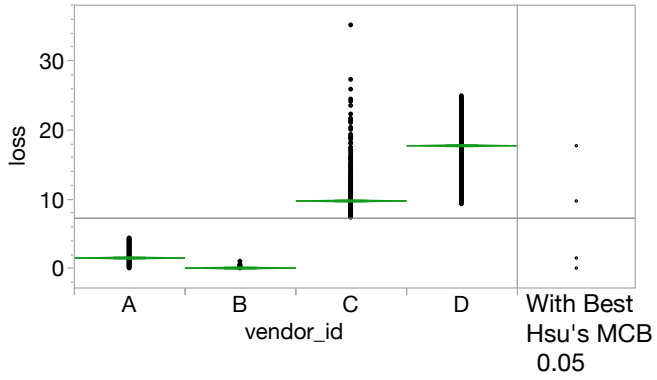


Figure 6.12: Partition tree for loss (quadratic loss using $\tau_1 = 10$) for the vendor selection example. $R^2 = 0.893$ after three splits (the maximum possible), resulting in a single vendor within each leaf.

Multiple comparisons with the best

A third approach described in Section 5.11.2 directly estimates the differences between each of the vendors and the apparent best. This multiple comparison approach controls the overall α for performing $k-1$ simultaneous tests. The means-diamonds plot for the vendor example appears in Figure 6.13a. We have such large sample sizes that the comparison circles look like dots. Vendor B clearly is the minimum loss choice.



(a) Means-diamonds plot of loss (quadratic loss for $\tau_1 = 10$) for four vendors.

Comparisons with Max and Min

Level	with Max p-Value	with Min p-Value
D	1.0000	<.0001*
C	<.0001*	<.0001*
A	<.0001*	<.0001*
B	<.0001*	1.0000

(b) p-values for MCB

Figure 6.13: Graphical and numerical depictions of loss for four vendors based on 1000 replications.

Subset selection

The subset selection procedure described in Section 5.11.2 yields a comparable conclusion. The subset consists solely of Vendor B.

6.3.3 Example: M/M/k queue

simulate it yourself

`QRunner.rb`
simulate a queue with either FIFO or LIFO processing rules, and the choice of several shapes for the service distribution.

Let's consider another stylized simulation example, based on simple M/M/k queues. Software for simulating these queues, as well as datasets used to generate the graphs, are available for your use. Note that we do not recommend the M/M/k assumptions for trying to address real-world problems—after all, how often have you encountered a queue where the most likely outcome is instantaneous completion of service? However, this is a useful exercise for demonstrating how robust design works for simulated systems because M/M queueing systems can be solved analytically, allowing us to confirm the validity of our simulation findings.

We will be evaluating the following four variants of an M/M/k queue, all with an arrival rate of 114 per hour:

A: a FIFO single server queue with service rate $\mu = 120$ per hour;

B: a LIFO single server queue with service rate $\mu = 120$ per hour;

C: a FIFO queue with two servers, each with service rate $\mu = 60$ per hour; and

D: a LIFO queue with two servers, each with service rate $\mu = 60$ per hour.

Our response is the customer time (difference between the customer's arrival time and the time they start service), and the target value is zero—unreachable, but intended to motivate minimizing wait times. In this case, our quadratic loss function still works even though we really have a one-sided loss. We do not need to explicitly provide a one-sided loss function because it is impossible for y to be below the target. We could think of this as a single decision factor called "Alternative" with four levels (A, B, C, and D). Alternatively, if we think of the alternatives as arising from crossing two factors ("queue discipline" and "number of servers"), our metamodels can be more informative.

We will fix several other input values for this experiment. Since the arrival rate is 114 per hour, we will set the overall service capability to 120 per hourt, yielding a traffic intensity of $\rho = 0.95$. This means that each of the n_{serve} servers can process $120/n_{serve}$ customers per hour on average. We will set the cost conversion constant $c = 1$ in the calculations of loss, which is commonly referred to as the *scaled loss*.

Let's conduct a small experiment. In the first replication, for each of the four alternatives, we make a run that stops after 1,000,000 customers have been served. The simulation reports the average, standard deviation, and maximum of the customer wait times, as well

as the average quadratic loss. We make 20 replications of the experiment. We made very long runs to avoid complicating the example by having to deal with the initial transient, a.k.a. simulation warm-up time. Altogether, this is a total of $4 \times 20 = 80$ simulation runs, representing 80 million customers.

Figure 6.14(a) provides boxplots of the average wait times from the 20 replications for each alternative. This corresponds to a traditional means-based assessment of system performance. Note that even though the individual estimates obtained via simulation are each based on one million customers, there is still a noticeable amount of variation in the actual values. Based on this figure, a traditional analysis would assess that there are slight positive benefits to having LIFO vs. FIFO queues, as well as slight benefits to replacing a single fast server with two slower servers.

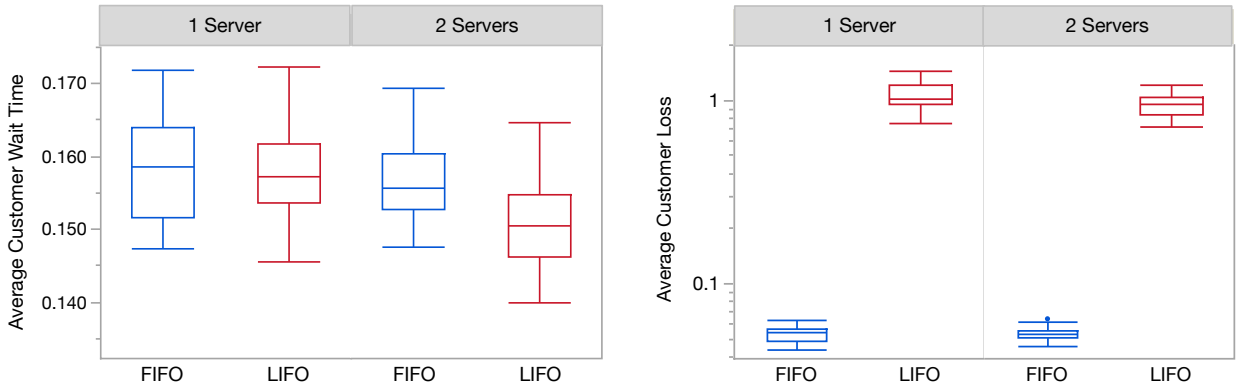


Figure 6.14: Average customer experience in four different queuing systems, (a) shows the average wait time and (b) shows the average loss from the customer perspective.

Figure 6.14(b) provides similar boxplots of the average customer losses, rather than the average customer wait times, with the vertical axis plotted on a log scale. The results are radically different with respect to queue discipline, showing that the average loss for the FIFO queues are negligible relative to those of the LIFO queues. Still, there appears to be a lower average loss associated with two slow servers rather than a single fast server. This differs from the analytical results obtained from queueing theory, which calculate steady-state mean customer waiting times of $0.158\overline{33}$ with one server and 0.1543 with two servers, regardless of queue discipline.

Consider the graph in Figure 6.15. This is based on data from the early portion of a longer simulation run. It shows the wait times for the first 310 customers arriving to an M/M/1 queueing system for two different queue disciplines: FIFO and LIFO. Common random numbers were used for each of the arrival and service processes to enable a head-to-head comparison between the two systems. The number 310 was chosen because that was the first time both queues were empty after at least 20 customers had been served.

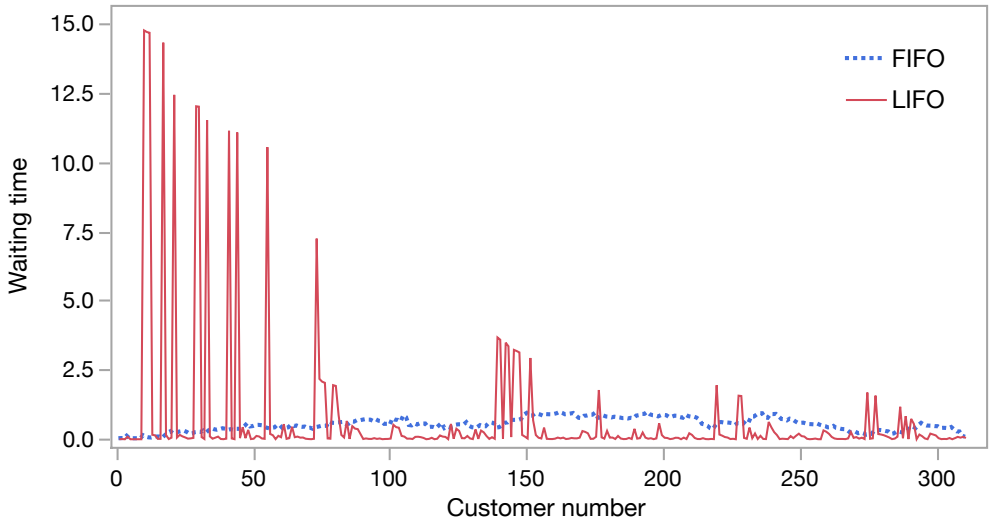


Figure 6.15: Wait times for the same group of 310 customers for two M/M/1 queues: one FIFO, one LIFO.

Some interesting statistics for selected customers from those in Figure 6.15 follow.

- Avg(wait)=0.0543 and Avg(loss)=0.0078 for the first 20 to arrive (or exit) the FIFO queue.
- Avg(wait)=0.0449 and Avg(loss)=0.0050 for the first 20 to exit the LIFO queue.
- Avg(wait)=2.9653 and Avg(loss)=43.0258 for the first 20 to arrive at the LIFO queue.

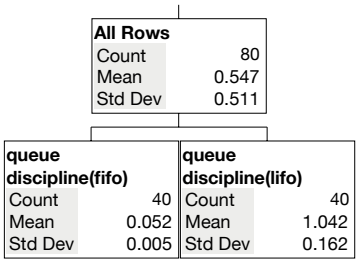
When we stop the simulation after a fixed number of customers have left the system, we are more likely to stop when the system has some customers waiting for service than when the system is empty and idle. Consequently, we are more apt to censor data from those who have long wait times, biasing our results downwards from the theoretical mean. This answers the thought question.

Queueing theory tells us that the expected wait time is the same for FIFO and LIFO systems having the same number of servers and the same traffic intensity. So what's going on here? A LIFO system serves a lot of customers faster than they would be served by a FIFO queue, but it does so by bottling up earlier arrivals behind later ones, and hence subjecting the backed-up customers to extremely long waits—there is a large amount of variability in the wait times. It averages out, but most people would consider the LIFO outcomes to be very unfair because the cost of waiting is placed heavily on the subset of bottled-up customers. This inequity is hidden if the analysis involves only the means. However, the quadratic loss function exposes it by explicitly including variance as part of the loss experienced by the population of customers as a whole.

Metamodeling

Now we return to the comparison of the four queueing systems. The graphs in Figure 6.14(b) are compelling, showing that the FIFO alternatives yield orders of magnitude lower loss than the LIFO alternatives. This is more informative than simply stating that alternatives A and C have lower loss. We may similarly find it more informative to use two factors with two levels each (the number of servers and the queue discipline) instead of a single factor with four levels obtained by crossing them, when fitting metamodels and quantifying the statistical significance. For brevity, we provide results for just two metamodels that directly fit the average loss: a partition tree metamodel and a multiple regression metamodel.

The partition tree metamodel appears in Figure 6.16. $R^2 = 0.95$ after a single split. Splitting twice more (the maximum possible for this study) is not statistically significant and has essentially no effect on R^2, which increases to 0.251.



(a) Partition tree.

Column Contributions

Term	Number of Splits	SS	Portion
queue discipline	1	19.609	1.0000
#servers	0	0.000	0.0000

(b) Column contributions.

Figure 6.16: Partition tree metamodel of quadratic loss ($\tau = 0$) for four different queueing systems. $R^2 = 0.950$ after a single split.

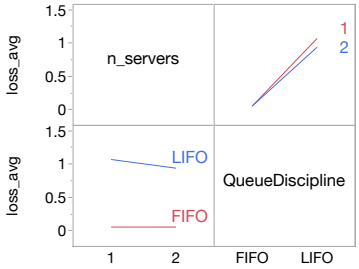
Regardless of whether you use the full tree or the tree of Figure 6.16, queue discipline has a very very strong (VVS) relative impact, while the number of servers has no impact because it fails to meet the threshold of 0.015625 for very weak (VW) relative impact.

The multiple regression metamodel results are shown in Figure 6.17. Here, the queue discipline is again, by far, the dominant factor, with LIFO systems resulting in substantially

Sorted Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
QueueDiscipline[FIFO]	-0.474482	0.013252	-35.81	<.0001*
(n_servers-1.5)*QueueDiscipline[FIFO]	0.0654149	0.026504	2.47	0.0158*
n_servers	-0.065197	0.026504	-2.46	0.0162*

(a) Sorted parameter estimates.



(b) Interaction plot.

Figure 6.17: Metamodel of quadratic loss ($\tau = 0$) for queueing systems differentiated by two 2-level factors. $R^2 = 0.945$ for the metamodel involving both main effects and their interaction.

higher losses. Both the interaction term and the number of servers are statistically significant at the $\alpha = 0.02$ level, although these may not be of practical importance.

We remark that care should be taken in interpreting any of these results, since the two-server systems have slower servers than the single-server system. In particular, they should not be interpreted as indicating that adding a second server to an existing system would provide little benefit.

6.4 Quantitative factors

We now move from qualitative factors, where we have a limited number of potential alternatives, to situations involving one or more quantitative factors, where there could be an infinite number of alternatives.

simulate it yourself

`1Dcurve.rb`
simulates the deterministic curve shown in Figure 6.18, which involves a single, continuous-valued decision factor x and a single, continuous-valued noise factor w.

6.4.1 Visual insights for a single-factor, stylized example

Initially, let's consider a response which is a deterministic function of x, as in Figure 6.18, i.e., there is no noise present. We can see graphically how y_{x_1} is determined by x_1, where the value of x_1 has been chosen to minimize the response.

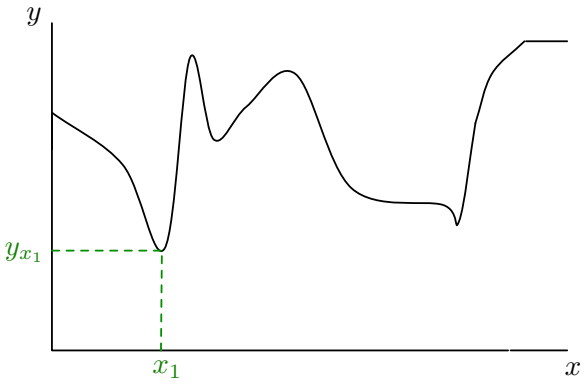


Figure 6.18: Deterministic response as a function of a single quantitative decision factor x. The (x_1, y_1) pair shown corresponds to the x-value that yields the global minimum of y.

Now let's suppose the target happens to be set at that minimum achievable response value, as in Figure 6.19. For this response surface, there is only one y_x that hits the target: it is represented by the green arrow. This would be the suggested operating condition, i.e., the choice of the decision factor's value. The orange arrows indicate choices of x that yield local but not global minima of y. They might be places where an optimization method such as gradient search gets "stuck."

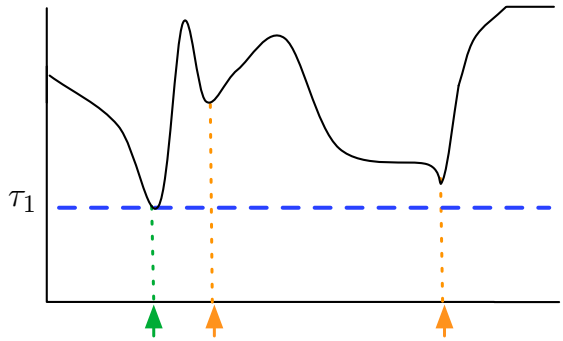


Figure 6.19: Considering τ_1 for a deterministic response with no noise, there is a single no-loss alternative.

When there is no noise present, then any value of x which for which $y_x = \tau$ is both an optimal solution and a robust solution. Figure 6.20 shows the same response surface, but with a different target value. In this situation, the decision maker has four potential optimal solutions. Having multiple options can be helpful if there are key considerations that may influence the final decision but have not been coded into the simulation model, such as the cost or ease of implementing the results. (We will discuss the cost of implementation in more detail in a later section.)

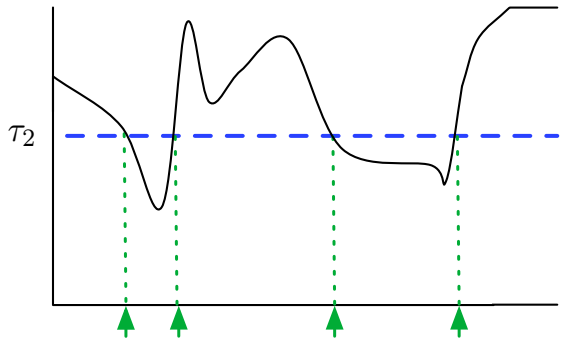


Figure 6.20: Considering τ_2 for a deterministic response with no noise, there are four no-loss alternatives.

Finally, there are cases where the target value is not achievable, as in Figure 6.21. You can think of this as an "ideal" value that is currently out of reach, or as an ambitious goal given the current situation. Here, the best (i.e., low loss) alternative would be to select the x that

yields the global minimum. While this is the same *decision* as that made in Figure 6.19, there is a non-zero loss associated with this decision. We remark that this does not mean the target value is wrong—advantages of a robust design approach are that the average loss is a meaningful way of measuring the quality of the solution, and that this can serve as an impetus for continual improvement efforts.

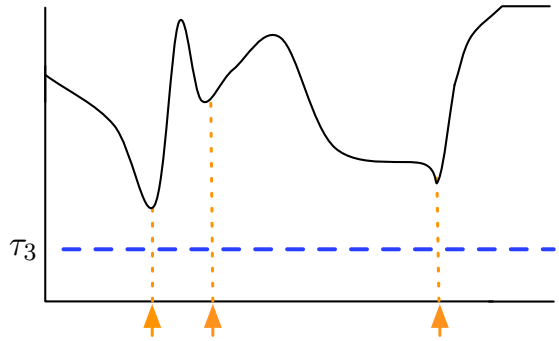


Figure 6.21: Considering τ_3 for a deterministic response with no noise, there are not any no-loss alternatives. One alternative qualifies as having the lowest loss.

Now let's see what happens with a specific type of noise—namely, although we have chosen a specific value of x, we may not be able to achieve that exact input value in practice. It is easy to think of ways this happens in the real world. One example might be that we're building a circuit and want a nominal 5-ohm resistor or 15-ohm resistor, but resistors themselves are not perfect. The actual resistance may be 5 ± 0.01 or 15 ± 0.01 ohms. Another might be related to testing the energy consumption of an electric vehicle, where operators are given a nominal speed but their actual speed will vary over the course of the test drive. A similar situation for our notional response surface is shown in Figure 6.22, where attempting to select a particular value of x leads to a value falling within $x \pm \Delta$. There are Two such intervals.

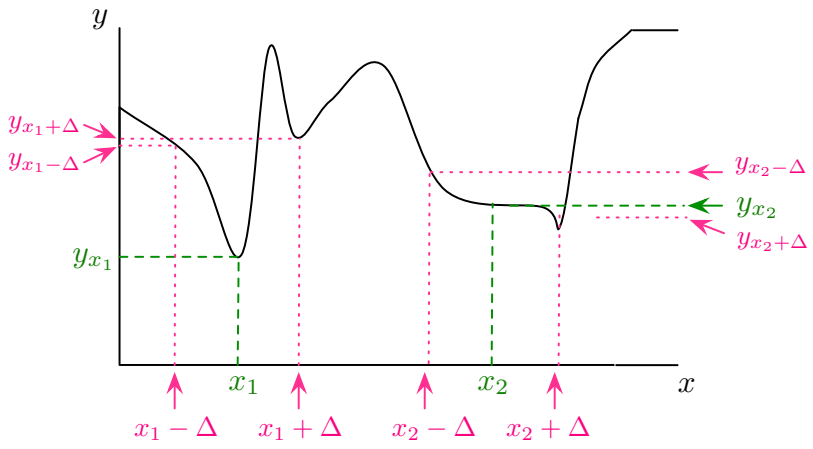


Figure 6.22: Values of y that result from the extremes of noisy settings of two values of x.

Note that y_{x_1} does not fall between $y_{x_1-\Delta}$ and $y_{x_1+\Delta}$. In fact, we need to sweep out the entire response ranges corresponding to $x \pm \Delta$ to get a sense of the robustness. Figure 6.23 does so. It shows that the variability around x_1 is amplified by the steep gradients in this region of the system, while the variability around x_2 is dampened by the system. Visually, x_2 has the potential to be more robust, depending on the target, by virtue of having less variability.

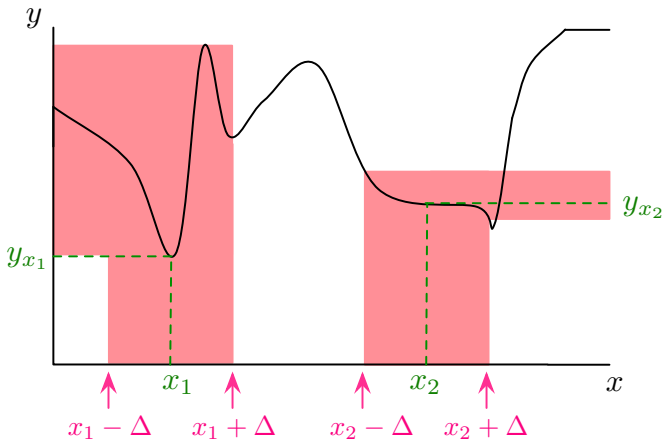


Figure 6.23: Values of y that result from the x values between $x_1 \pm \Delta$ and $x_2 \pm \Delta$.

In general, we do not know the underlying curve as we do in this artificial example, but instead are trying to evaluate the implicit I/O relationship from our simulation model.

This view of loss is very different from the additive, constant-variance error models we often assume, either consciously or unconsciously. Figure 6.24 shows what the variability in y looks like given that classical regression assumption—regardless of the value of x, the values of y range lie within $y_x \pm \delta$ for some constant δ.

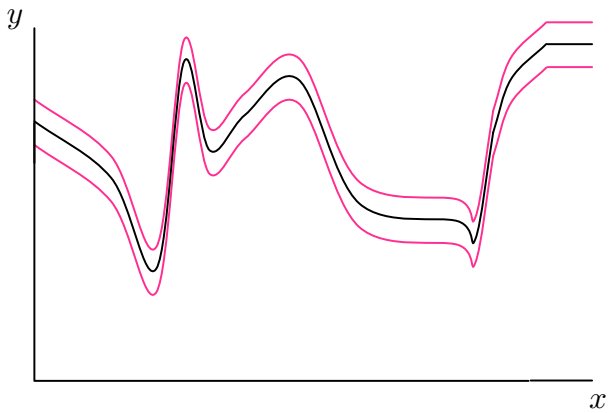


Figure 6.24: An assumption of constant-variance additive noise might reflect intrinsic noise associated with y, but does not convey the loss associated with uncertainty in x.

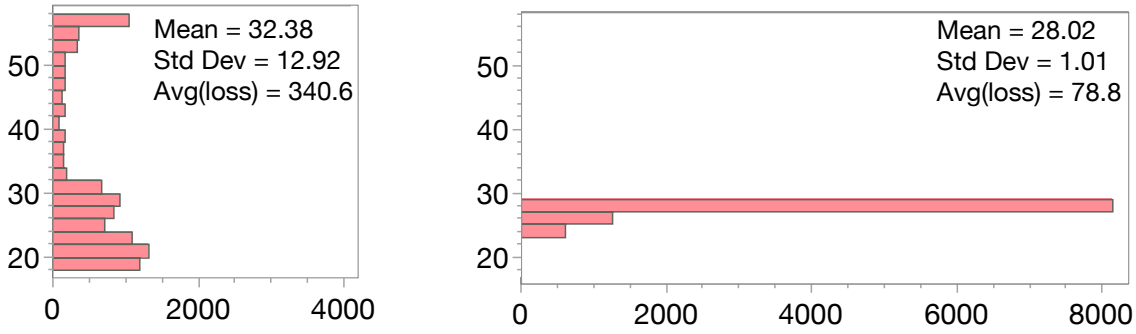
A few other points should be noted:

- Stochastic systems may have intrinsic sources of variability, and that variability may depend on x, i.e., the variability may be non-homogeneous. Any intrinsic variability which is present will also contribute to loss.
- For ease of illustration, we assumed the noise around x was uniformly distributed over a constant-width range. This is not a necessary assumption for robust design.
- With multiple factors, each factor can have have its own distributional behavior.

6.4.2 Numerical details for a single-factor, stylized example

We will continue with the same model presented in Section 6.4.1, but now we'll quantify the results. As we move from graphical intuition to numerical analysis, we need to formally define the ranges involved since Figures 6.18 through 6.24 did not have scales on the axes. For this model the x values go from 0.0 to 1.0 and the y axis scale goes from 0.0 to 60.0. The two points x_1 and x_2 have values 0.2 and 0.67, respectively. The y values associated with these two points are $y_{x_1} = 19.2$ (the global minimum) and $y_{x_2} = 28.7$.

If one were not aware of DOE, then a reasonable approach for estimating $E[Y]$ for the stochastic version of the model would be to randomly generate values from the input distributions around x_1 and x_2. Figure 6.25 shows histograms that result from 10,000 y outputs obtained from uniform sampling over the input intervals $x_1 \in 0.21 \pm 0.6$ and $x_2 \in 0.72 \pm 0.6$. These correspond to the range of values swept out in Figure 6.23. Neither of these distributions is symmetric or bell-shaped. The Avg(loss) values are calculated from the sample means and standard deviations using a target $\tau = 19.2$ and a cost coefficient of 1.0.



(a) Decision factor value is $x = 0.20$.

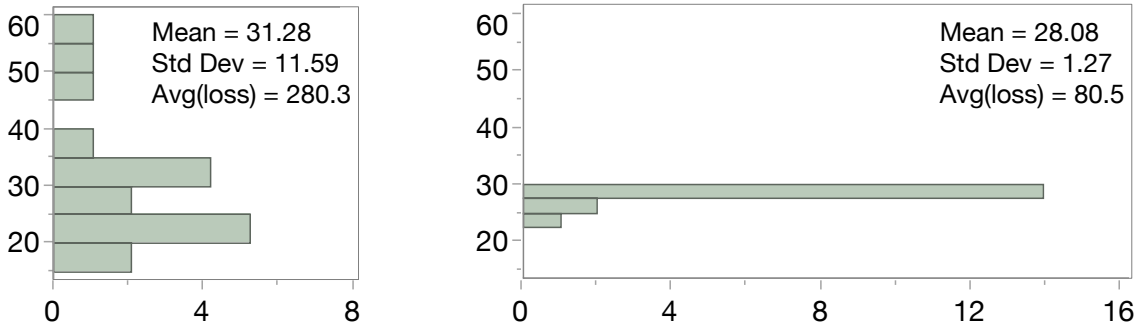
(b) Decision factor value is $x = 0.72$.

Figure 6.25: Histograms of y, each based on 10,000 random Uniform samples in $x \pm 0.06$. The same bins are used for (a) and (b).

In summary, if we ignored variability, we would select x_1 rather than x_2 and expect to incur no loss. However, if we implemented the two alternatives, then the noise variation would impact the results. We would achieve an (estimated) loss of 340.6 with x_1, compared to a

loss of only 78.8 with x_2. In other words, by selecting x_1 instead of x_2 we would incur a loss over four times as large! This would be particularly shocking to decision makers if we had incorrectly recommended x_1 as a perfect solution.

We can get very similar information much more efficiently by considering the variation around the nominal x as a noise factor that we explicitly control in the virtual world, although it may be uncontrollable in the real-world setting. A simple way of accomplishing this, in general, is to cross a design for the decision factors (denoted by X) with a design for the noise factors (denoted by W). For this small example, we have a single factor of each type. For illustrative purposes, we focus just on the values $x_1 = 0.20$ and $x_2 = 0.72$ values in Figure 6.23 as a 2-level design for the decision factor. Crossing this with a 17 dp NOLH for the noise factor w yields a total of $2 \times 17 = 34$ dps. We can take a single replication because there is no intrinsic noise. Histograms and summary statistics of a single replication of the results are shown in Figure 6.26. The summary statistic and Avg(loss) calculations are somewhat different, particularly for x_1, but still indicate that x_2 is a much more robust solution than x_1: the Avg(loss) for x_1 is 3.5 times as large as that for x_2. However, using this simple design for our experiment let us arrive at a good decision in 34 runs rather than 20,000 runs.



(a) Decision factor value is $x = 0.20$. **(b)** Decision factor value is $x = 0.72$.

Figure 6.26: Histograms of y, each based on a 17-dp NOLH noise factor design with levels ranging over $w = (-0.06, 0.06)$.

6.4.3 General approach for multiple factors

By far the more interesting situation for identifying robust solutions involves multiple factors. When one or more of these factors are quantitative, then metamodels can be used very effectively.

Setting up the experiment

When planning a robust experiment, you should first categorize each of the potential factors in the experiment as a decision factor or a noise factor. There are two basic ways of constructing a design that involves both types of factors.

- Cross separate designs: If D_D is the design for the decision factors and D_N is the design for the noise factors, then $D_D \times D_N$ is the crossed design.
- Create a combined design: No distinction is made among factors when creating the design, the distinction is only applied when performing the analysis.

Each method has its pros and cons. One benefit of the crossed design is the ease of explaining the concept. Since each decision factor dp is subjected to the same combinations of noise factors, it's possible to make "apples to apples" comparisons for simple descriptive statistics and graphics. If you later decide that some of the noise factor dps are not suitable, it's easy to remove these from the experiment. Another benefit of using a separate design for noise factors is that we need not select a design as capable of estimating high-order effects or interactions without confounding. Interaction and quadratic terms among noise factors are not of interest, we are solely concerned about how the noise is transmitted through to induce variability in the overall response. Consequently, we can use a simpler or smaller design for the noise factors than for the decision factors. For example, we might choose a resolution 3 fractional factorial or CCD, or a single stack of the smallest NOLH capable of handling these factors for the noise factor design. For the decision factor design, a larger NOLH or FBS or FBD will ensure we have enough metamodeling flexibility for identifying important decision factor effects and interactions, for lack-of-fit indications, and for graphical methods that do a better job of revealing relationships when the designs are have better space-filling characteristics. With a crossed design, it is also easy to expand or contract the noise design if necessary, e.g., by adding other factors, removing dps that are later determined to be not realistic or not of interest, or adding additional levels for a qualitative noise factor. The major drawback of a crossed design is its size.

A combined design also offers several benefits. One is that it can be much more efficient than a crossed design. For example, with 20 decision factors and 20 noise factors, crossing two NOLHs would require $256 \times 256 = 65,536$ dps—over 500 times larger than using a 128-dp NOAB as a combined design. The potential computational savings are high, especially if the simulation takes a long time to run. We cannot easily add factors to combined designs unless we resort to crossing, which is a potential drawback. Combined designs must be analyzed in the metamodel domain—we cannot make "apples to apples" comparisons. This can be a drawback for communicating with some decision makers. However, it can also be a benefit if the decision makers decide that a factor originally categorized as a noise factor should be treated as a decision factor, or vice versa, because you can update the analysis without having to create and run a new experiment.

Generic situations when using a dual metamodel approach

Table 6.2 is a multi-page presentation of several illustrative scenarios. In each scenario, separate metamodels are constructed for mean and variance, and advice is given for how to achieve the stated scenario goal with minimal quadratic loss.

Table 6.2: Determining robust solutions from separate metamodels for response mean and variability using a quadratic loss function with scaling constant $c = 1$. Several illustrative scenarios are provided below, each involving 10 continuous-valued decision factors with the following ranges: $x_i \in [1, 10]$ for $i = 1, \ldots, 3$, $x_i \in [0, 10]$ for $i = 4, \ldots, 10$. We further assume that the cheapest or most convenient levels for each factor correspond to their lowest levels.

Scenario	**Details**
Serendipity	Changing any important factor that appears in both metamodels and is either good for both, or bad for both.
Example:	
Goal and Target:	Make μ as small as possible (note: $\tau = 0$ if we assume $\mu \geq 0$).
Metamodels:	$\mu \approx 54 - 3x_1 - x_2 - 0.1x_1x_2$ $\sigma \approx 10 - 0.1x_1 - 0.2x_2$
Solution Approach:	Make both x_1 and x_2 large to decrease both the mean and the variability. Set other factors to their cheapest or most convenient levels.
Decision:	$x_1 = 10$, $x_2 = 10$, $x_3 = 1$, $x_4 = x_5 = \ldots = x_{10} = 0$. This robust solution also achieves the best mean and the lowest variance.
Average Loss:	$(4 - 0)^2 + 7^2 = 65$
Calibration	We can calibrate the mean while minimizing the variability.
Example 1:	
Goal and Target:	Make μ as close to $\tau \equiv 100$ as possible.
Metamodels:	$\mu \approx 42 + 3x_1 + x_2 + 0.25x_1x_2$ $\sigma \approx 10 - 0.5x_1 + x_3$
Solution Approach:	Make x_1 large and x_3 small to minimize variability, then solve for x_2 to achieve the target value. Set other factors to their cheapest or most convenient levels.
Decision:	$x_1 = 10$, $x_2 = 8$, $x_3 = 1$, $x_4 = x_5 = \ldots = x_{10} = 0$. This robust solution also achieves the best mean and the lowest variance.
Average Loss:	$(100 - 100)^2 + 6^2 = 36$
Example 2:	
Goal and Target:	Make μ as close to $\tau \equiv 100$ as possible.
Metamodels:	$\mu \approx 42 + 3x_1 + 4x_2 + 2x_3 + 0.5x_1x_3$ $\sigma \approx 10 - 0.5x_1 + x_3$
Solution Approach:	Make x_1 large and x_3 small to minimize variability, then $\mu \approx 42 + 30 + 4x_2 + 2 + 0.5(10)(1) = 79 + 4x_2$. Solve for x_2 to achieve the target value. Set others to their cheapest or most convenient levels.
Decision:	$x_1 = 10$, $x_2 = 4.25$, $x_3 = 1$, $x_4 = x_5 = \ldots = x_{10} = 0$. This robust solution also achieves the best mean and the lowest variance.
Average Loss:	$(100 - 100)^2 + 6^2 = 36$
Tradeoffs	We must make trade-offs among the mean and variability.
Example:	
Goal and Target:	Make μ as close to $\tau \equiv 500$ as possible.
Metamodels:	$\mu \approx 458 + x_1 + 2x_2$ $\sigma \approx 1 + 2x_1$

Continued on next page

Table 6.2 – Continued from previous page

Scenario	**Details**
Solution Approach:	We cannot achieve the target mean while setting x_1 to its lowest value, when $x_1 = 1$ we have $\mu \approx 458 + 1 + 2x_2$ which tops out at $\mu \approx 479$. Because increasing x_1 also increases the variability we must work in the (scaled) loss domain to minimize $E[Loss]$, using gradient methods or a numerical (grid) search if necessary. For this particular example, we notice that making x_2 as large as possible gets us closest to the target without a detrimental impact on variability. $E[Loss] = (478 + x_1 - 500)^2 + (1 + 2x_1)^2 = 5x_1^2 - 40x_1 + 485$. The derivative with respect to x_1 is $10x_1 - 40$, setting this to zero and solving yields $x_1 = 4$. Set other factors to their cheapest or most convenient levels.
Decision:	$x_1 = 4$, $x_2 = 10$; $x_3 = 1$, $x_4 = x_5 = \ldots = x_{10} = 0$. This robust solution with $\mu \approx 482$ and $\sigma \approx 9$ is neither the best mean nor the lowest variance solution.
Average Loss:	$(482 - 500)^2 + 9^2 = 405$
Means only	The factors impact only the mean.
Example 1:	
Goal and Target:	Make μ as close to $\tau \equiv 110$ as possible.
Metamodels:	$\mu \approx 40 + 3x_1 + 4x_2$ $\sigma \approx 8$
Solution Approach:	Setting the metamodel equation for $\mu = 110$, we find there is only one solution: $x_1 = x_2 = 10$. Set other factors to their cheapest or most convenient levels.
Decision:	$x_1 = 10$, $x_2 = 10$; $x_3 = 1$, $x_4 = x_5 = \ldots = x_{10} = 0$.
Average Loss:	0
Example 2:	
Goal and Target:	Make μ as close to $\tau \equiv 110$ as possible.
Metamodels:	$\mu \approx 54 + 3x_1 + 4x_2$ $\sigma \approx 8$
Solution Approach:	Setting the metamodel equation for $\mu = 110$, we find there are multiple low-loss solutions: $x_2 \in [6, 10]$ and $x_1 = 18 - 4x_2/3$. Set other factors to their cheapest or most convenient levels.
Decision:	$x_2 \in [6, 10]$ and $x_1 = 18 - 4x_2/3$; $x_3 = 1$, $x_4 = x_5 = \ldots = x_{10} = 0$.
Average Loss:	0
Variability only	The factors impact only the variability.
Example 1:	
Goal and Target:	Make μ as close to $\tau \equiv 75$ as possible.
Metamodels:	$\mu \approx 79$ $\sigma \approx 14 - x_1 + x_2$
Solution Approach:	The mean does not depend on the factors, but the loss certainly does. Make x_1 large and x_2 small, set the other factors to their cheapest or most convenient levels.

Continued on next page

Table 6.2 – Continued from previous page

Scenario	**Details**
Decision:	$x_1 = 10$, $x_2 = 1$; $x_3 = 1$, $x_4 = x_5 = \ldots = x_{10} = 0$.
Average Loss:	$5^2 + (79 - 75)^2 = 41$
No impact	The factors do not impact either the mean or the variability.
Example 1:	
Goal and Target:	Make μ as close to $\tau \equiv 110$ as possible.
Metamodels:	$\mu \approx 90$ $\sigma \approx 12$
Solution Approach:	There is nothing to solve, because none of your factors appear to affect response distribution. This doesn't mean your system is perfectly predictable—the level of variability may be high or low. However, all alternatives investigated seem equally good (or equally bad). First, confirm that there are no bugs in your program and that your data farming wrapper is working properly. Consider metamodels other than polynomial forms. If you still find no significant causal relationships, set all factors to their cheapest or most convenient levels, or reexamine your assumptions to see if it is more informative to either (i) change the factor levels or (ii) explore more factors.
Decision:	Set all factors at their lowest cheapest or most convenient levels.
Average Loss:	$(90 - 110)^2 + 12^2 = 544$

These scenarios may help you make use of the two metamodels to identify robust solutions. Of course, if the apparent best solution is not one that you have already tested, then making confirmation runs is necessary before you make recommendations or attempt a real-world implementation based upon your findings. This is another benefit we have in simulation settings that may not be available to those running physical experiments, as discussed in Section 4.8.5. However, we need to extend the idea of confirmation runs in the robust design settings, by crossing any decision factor dps of interest with a noise factor design, and then summarizing the results over the decision factor dps to see if the resulting mean and variance are sufficiently close to the metamodel predictions.

Following the PDSA cycle in Chapter 1, if we confirm that we have found robust solutions our next action can be to pass these findings up the chain. Alternatively, our exploration might identify a need to conduct additional experiments—perhaps changing the type of design, the sets of factors explored, or the factor ranges—so we can refine our metamodels until they are satisfactory.

Appendices

Appendix A

Installing and using Ruby and gems

Many of the tools and demos discussed in this book are written in the Ruby programming language. In order to "follow along", you will need to have Ruby installed. As it says in "The Hitchhiker's Guide to the Galaxy," DON'T PANIC!!! You do not need to program anything in Ruby, any more than you need to know C# to use Excel or Word. Gems are Ruby's package distribution mechanism, and after they're installed they should be fully available from the command-line in any working directory.

For the most up-to-date instructions, please look at the README file found at `https://bitbucket.org/paul_j_sanchez/datafarmingworkshop/`. OS-specific videos are available in the `SoftwareSetup` folder of that archive site. They may not be as up to date as the written instructions, but you should find them helpful regardless.

Updates to the datafarming gem (and any other gems on your system) can be performed at any time after the initial installation by typing the command "`gem update`" (without the quotes) in a terminal/command window.

Appendix B

Common Distributions

Throughout this Appendix we will follow the common convention of using upper case letters such as X or Y to represent random variables, and lower case letters such as x or y to signify particular outcomes or values. The notation $Y \sim N(\mu, \sigma^2)$ indicates that the r.v. Y is distributed Normally with mean μ and variance σ^2; $Y \sim U(a, b)$ indicates that the r.v. Y is distributed according to a Uniform(a,b) distribution, etc. We also assume familiarity with standard mathematical symbology such as $\forall x$ ("for all x"), $A \iff B$ ("A if and only if B"), and summation and integration notation.

This Appendix is intended as a refresher for those without recent study of statistics. We show the distributions graphically, and indicate how to find certain statistics using R, JMP, or as examples from tabled values.

R / RStudio Commands

We assume that you are able to read in data to RStudio. In what follows, we provide the R commands that should be typed into your RStudio console window. By convention, we will use

```
blue text in Courier font
```

to indicate an R command.

JMP Navigation

We assume that you are able to read in data to JMP. In what follows, we provide the JMP navigation steps that should be followed to carry out the tasks. By convention, we will use *purple italic text* to indicate the menu items, with arrows indicating sub-menu choices and purple non-italic when using formulas. E.g.,

Analyze → *Distribution*

means select *Analyze* at the top menu, then *Distribution* from its submenu.

This Appendix is organizes as follows. In Section B.1 we describe several distributions from which p-values or quantiles are used for testing hypotheses and constructing confidence intervals. In Section B.2 we summarize the central limit theorem. In Section B.3 we provide short descriptions of several distributions used as sources of randomness in the reference simulation models.

B.1 Distributions and Critical Values

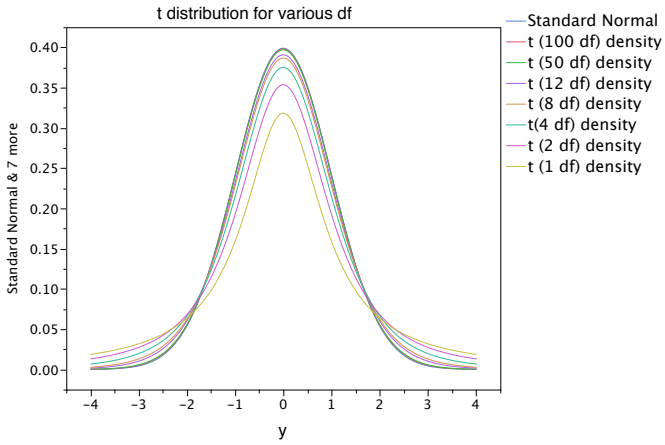
t Distribution

Overview: The t distribution with ν degrees of freedom is the probability distribution you get when you divide a standard Normal $\mathrm{N}(0,1)$ r.v. by an (independent) χ^2 r.v. with ν degrees of freedom. This may sound esoteric, but is very common in practice.

Fun facts:

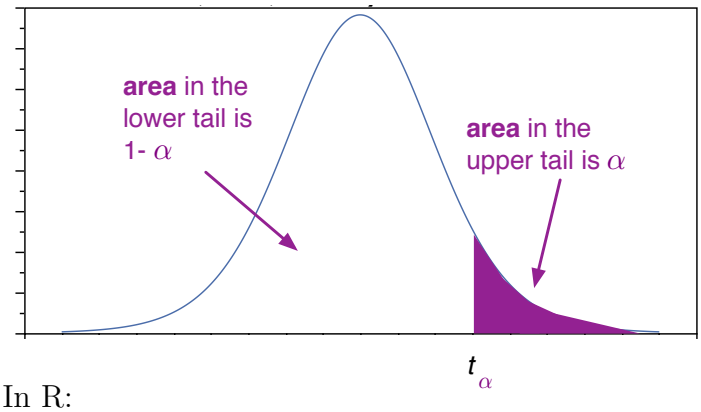
- If Y has a t distribution with ν d.f., then:
  - $E[Y] = 0$, $\mathrm{Var}(Y) = \frac{\nu}{\nu-2}$ for $\nu > 2$, skewness $= 0$
  - The pursuit of brewing a better beer led to its discovery....
  - The t distribution is symmetric and bell-shaped, but more "squashed" than the Normal— especially if d.f. are low.
- (Definition 7.2) Let Z be a standard normal r.v. and W be a χ^2_ν r.v. Then if Z and W are independent,
$$T = \frac{Z}{\sqrt{W/\nu}} \sim t_\nu$$
where t_ν is the t distribution with ν d.f.
- In particular, note that:
$$\frac{\overline{Y} - \mu_{\mathrm{Y}}}{S_Y/\sqrt{n}} \equiv \frac{\overline{Y} - \mu_{\overline{\mathrm{Y}}}}{S_{\overline{Y}}} = \frac{\left(\overline{Y} - \mu_{\overline{\mathrm{Y}}}\right)/\sigma_{\overline{Y}}}{S_{\overline{Y}}/\sigma_{\overline{Y}}} = \frac{Z}{\sqrt{(S_Y^2/n)/(\sigma_Y^2/n)}} = \frac{Z}{\sqrt{\frac{(n-1)S_Y^2}{\sigma_Y^2}/(n-1)}} \sim t_\nu$$
We'll often end up calculating this to "normalize" data, and figure out "how far away" (in terms of standard deviations) our sample mean is from the population mean μ_{Y}.
- As ν becomes large, the t distribution converges to the standard Normal.

Pictures: This figure shows several t distributions with different d.f.



Calculating critical values: Most often, if we're looking up values of the t distribution we'll be interested in specific percentiles. Once again, a quick sketch will help you keep straight whether you're looking for an upper tail, lower tail, etc.

★ Tabled information: A short snippet of a typical upper-tailed table is shown below.



df	...	$t_{0.100}$	$t_{0.050}$	...
1	…	3.078	6.314	…
2	…	1.866	2.920	…
3	…	1.638	2.353	…
4	...	1.533	2.132	...
infinity		1.282	1.645	

★ In R:

To calculate $\Pr\{Y \leq y\}$ for $Y \sim t_\nu$, then the command to use is:

```
pt(y, ν)
```

To calculate $\Pr\{Y > y\}$ for $Y \sim t_\nu$, then use:

```
pt(y, ν, lower.tail = FALSE)
```

To calculate q so that $\Pr\{Y \leq q\} = p$ for $Y \sim t_\nu$ and some $0 \leq p \leq 1$, then use

```
qt(p, ν)
```

To calculate q so that $\Pr\{Y > q\} = p$ for $Y \sim t_\nu$ and some $0 \leq p \leq 1$, then use:

```
qt(p, ν, lower.tail = FALSE)
```

★ In JMP:

To calculate $\Pr\{Y \leq y\}$ for $Y \sim t_\nu$, right-click on a blank column, select Formula and then Probability→t Distribution and fill in the numbers to get

t Distribution(y, ν)

To calculate the upper tail area, do this with the formula

$1 -$ (t Distribution(y, ν))

To calculate q so that $\Pr\{Y \leq q\} = p$ for $Y \sim t_\nu$ and some $0 \leq p \leq 1$, use

t Quantile(p, ν)

To calculate q so that $\Pr\{Y > q\} = p$ for $Y \sim t_\nu$ and some $0 \leq p \leq 1$, use

t Quantile(1-p, ν))

Common uses: The t distribution is used for several things, but the most common are:

- Constructing confidence intervals or conducting hypothesis tests about a mean
- Constructing confidence intervals or conducting hypothesis tests about differences between two means

χ^2 (Chi-squared) Distribution

Pronunciation: The Greek letter χ (chi) is pronounced "ky" and the Greek letter ν (nu) is pronounced "new."

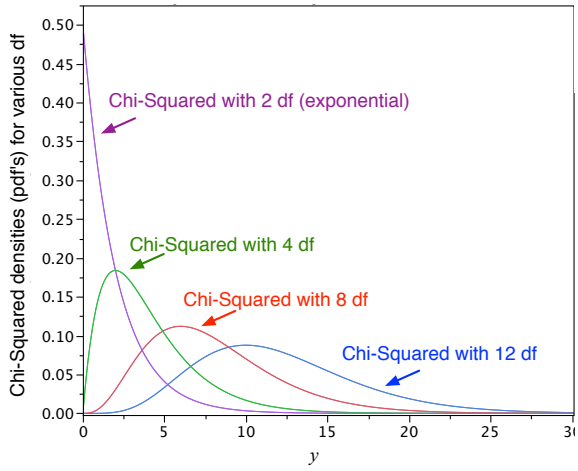
Overview: The χ^2 distribution with ν degrees of freedom is the probability distribution you get when you add up ν squared standard normal $N(0,1)$ random variables. Its lowest possible value is zero, and (like the normal) its highest possible value is infinity. It is skewed to the right for small and moderate ν, though for large ν it is approximately $\mathrm{N}(\nu, 2\nu)$.

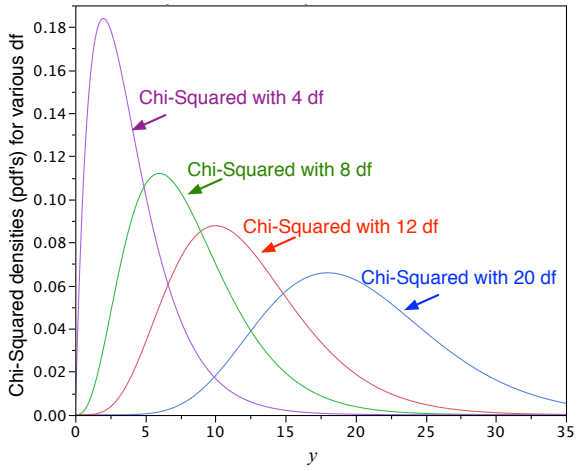
Fun facts:

- Let Y be a χ^2_ν random variable.
  - $E[Y] = \nu$, $\mathrm{Var}(Y) = 2\nu$, $\mathrm{skewness}(Y) = \sqrt{8/\nu}$
  - The χ^2_ν is a Gamma r.v. with parameters $\alpha = \nu/2$ and $\beta = 2$
  - As a special case, the χ^2_2 is an exponential r.v.
- Let $Y_1, \ldots, Y_n$ be iid observations from a $\mathrm{N}(\mu_Y, \sigma^2_Y)$ distribution. Then

$$\frac{(n-1)S^2}{\sigma^2_Y} \equiv \frac{1}{\sigma^2_Y}\sum_{i=1}^{n}\left(Y_i - \overline{Y}\right)^2 \sim \chi^2_{n-1}$$

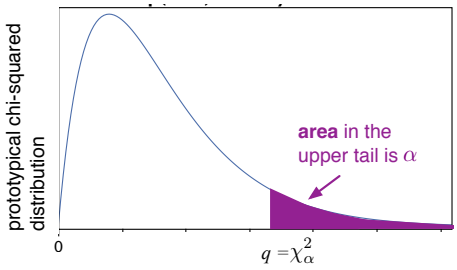
Pictures: The left figure shows pdfs of several different χ^2 distributions, including the exponential. The right figure has some of the same χ^2 distributions, but it makes it clearer that as the d.f. increase, the distribution starts to lose its skewness and converge toward a normal.





Calculating critical values: Most often, if we're looking up values of the χ^2 distribution we'll be interested in specific percentiles—either for the upper tail or lower tail (or both)! Different books and software use different notation, so sketching a quick picture will help you make sure you keep things straight.

★ Tabled information: A short snippet of a typical upper-tailed table is shown below.



df	...	$\chi^2_{0.950}$	$\chi^2_{0.900}$	...	$\chi^2_{0.100}$	$\chi^2_{0.050}$	...
1	...	0.0039321	0.0157908	...	2.70554	3.84146	...
2	...	0.102587	0.210720	...	4.60517	5.99147	...
3	...	0.351846	0.584375	...	6.25139	7.81473	...
4	...	0.710721	1.063623	...	7.77944	9.48773	...

★ In R:

To calculate $\Pr\{Y \leq y\}$ for $Y \sim \chi^2_\nu$, the command is:

`pchisq(y, ν)`

To calculate $\Pr\{Y > y\}$ for $Y \sim \chi^2_\nu$, the command is:

`pchisq(y, ν, lower.tail = FALSE)`

To calculate q so that $\Pr\{Y \leq q\} = p$ for $Y \sim \chi^2_\nu$ and some $0 \leq p \leq 1$, then use

`qchisq(p, ν)`

To calculate q so that $\Pr\{Y > q\} = p$ for $Y \sim \chi^2_\nu$ and some $0 \leq p \leq 1$, use

`qchisq(p, ν, lower.tail = FALSE)`

★ In JMP:

To calculate $\Pr\{Y \leq y\}$ for $Y \sim \chi^2_\nu$, right-click on a blank column, select Formula and then Probability→ChiSquare Distribution and fill in the numbers to get

ChiSquare Distribution(y, ν)

To calculate the upper tail area, do this with the formula

1 − (ChiSquare Distribution(y, ν))

To calculate q so that $\Pr\{Y \leq q\} = p$ for $Y \sim \chi^2_\nu$ and some $0 \leq p \leq 1$, use

ChiSquare Quantile(p, ν)

To calculate q so that $\Pr\{Y > q\} = p$ for $Y \sim \chi^2_\nu$ and some $0 \leq p \leq 1$, use

ChiSquare Quantile(1-p, ν)

Common uses: The χ^2 distribution is used for several things, including:

- Constructing confidence intervals or conducting hypothesis tests about a population variance;
- Goodness-of-fit tests to see whether or not the data appear to come from specific distributions;
- Tests of independence, to see whether or not two qualitative factors are related to each other.

F Distribution

Overview: The F distribution has two sets of degrees of freedom: those in the numerator and in the denominator. It arises when you take the ratio of two χ^2 r.v.s, scaled appropriately.

Fun facts:

- Named for Sir Ronald Fisher, one of the founders of statistics and a lover of tea
- The F distribution takes on non-negative values and is skewed right
- For large ν_1, ν_2 the F distribution converges to a point distribution right around 1
- The mean and mode are close to 1.0, especially for large ν_1 and ν_2 (there are analytic expressions for these and other summary measures)
- Let W_1 and W_2 be *independent* χ^2 r.v.s with ν_1 and ν_2 d.f., respectively. Then

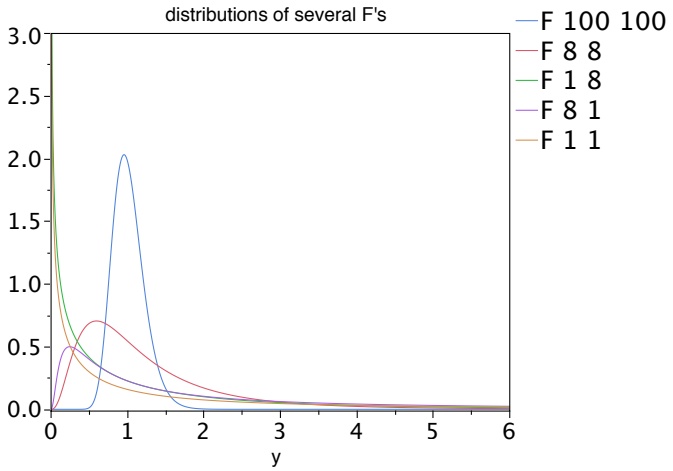
$$F = \frac{W_1/\nu_1}{W_2/\nu_2} \sim F_{\nu_1,\nu_2}$$

  where F_{ν_1,ν_2} is the F distribution with ν_1 d.f. in the numerator and ν_2 d.f. in the denominator.

- With a little algebra, we can show that:

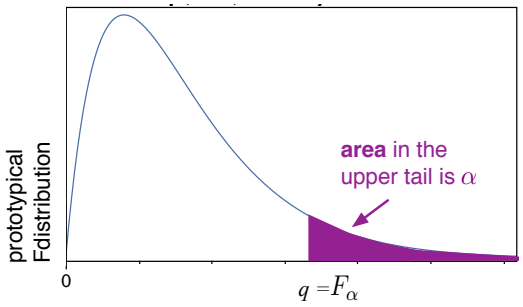
$$\frac{S_1^2/\sigma_1^2}{S_2^2/\sigma_2^2} \sim F_{\nu_1,\nu_2} \qquad \text{and} \qquad F_{\nu_1,\nu_2;\alpha} = \frac{1}{F_{\nu_2,\nu_1;1-\alpha}}$$

Pictures: This figure shows several F distributions with different sets of d.f. Note that the distribution is extremely skewed if there are low d.f. in the denominator.



Calculating critical values: Most often, if we're looking up values of the F distribution we'll be interested in specific percentiles. Once again, a quick sketch will help you keep straight whether you're looking for an upper tail, lower tail, etc.

★ Tabled information: A short snippet of a typical upper-tailed F table is shown below. Here, several values of α are shown for each combination of numerator and denominator d.f. Alternatively, separate tables may be given for each α.



F_α

Denominator df	Numerator df						
	α	1	2	3	4	5	…
8	.100	3.46	3.11	2.92	2.81	2.73	
	.050	5.32	4.46	4.07	3.84	3.69	
	.025	7.57	6.06	5.42	5.05	4.82	
	,010	11.26	8.65	7.59	7.01	6.63	
	.005	14.69	11.04	9.60	8.81	8.30	
9	.100	3.36	3.01	2.81	2.69	2.61	
	.050	5.12	4.26	3.86	3.63	3.48	
	.025	7.21	5.71	5.08	4.72	4.48	
	,010	10.56	8.02	6.99	6.42	6.06	
	.005	13.61	10.11	8.72	7.96	7.47	
⋮							

★ In R:

To calculate $\Pr\{Y \le y\}$ for $Y \sim F_{\nu_1,\nu_2}$, then the command to use is:

`pf(y, ν1, ν2)`

To calculate $\Pr\{Y > y\}$ for $Y \sim F_{\nu_1,\nu_2}$, then the command to use is:

`pf(y, ν1, ν2, lower.tail = FALSE)`

To calculate q so that $\Pr\{Y \le q\} = p$ for $Y \sim F_{\nu_1,\nu_2}$ and some $0 \le p \le 1$, then use

`qf(p, ν1, ν2)`

To calculate q so that $\Pr\{Y > q\} = p$ for $Y \sim F_{\nu_1,\nu_2}$ and some $0 \le p \le 1$, then use

`qf(p, ν1, ν2, lower.tail = FALSE)`

★ In JMP:

To calculate $\Pr\{Y \le y\}$ for $Y \sim F_{\nu_1,\nu_2}$, then right-click on a blank column, select Formula and then Probability→F Distribution and fill in the numbers to get

F Distribution(y, ν_1, ν_2)

To calculate the upper tail area, do this with the formula

1 − (F Distribution(y, ν_1, ν_2))

To calculate q so that $\Pr\{Y \le q\} = p$ for $Y \sim F_{\nu_1,\nu_2}$ and some $0 \le p \le 1$, use

F Quantile(p, ν_1, ν_2)

To calculate q so that $\Pr\{Y > q\} = p$ for $Y \sim F_{\nu_1,\nu_2}$ and some $0 \le p \le 1$, use

F Quantile(1-p, ν_1, ν_2))

Common uses: The F distribution is used for:

- Constructing confidence intervals or conducting hypothesis tests for the ratios of two variances

B.2 Central Limit Theorem

Overview: This is one of the most important analytic theorems in statistics—it lets people focus on a few key distributions when making inferences.

Definition: Let $Y_1, Y_2, \ldots, Y_n$ be iid r.v.s with $\mathrm{E}[Y_i] = \mu_Y$ and $\mathrm{Var}[Y_i] = \sigma_Y^2 < \infty$. Then as $n \to \infty$, the distribution of

$$\frac{\overline{Y} - \mu_Y}{\sigma_Y/\sqrt{n}}$$

converges to a standard Normal distribution .

Other equivalent ways of writing or thinking of the result:

- (without subscripts) Let $Y_1, Y_2, \ldots, Y_n$ be iid r.v.s with $\mathrm{E}[Y_i] = \mu$ and $\mathrm{Var}[Y_i] = \sigma^2 < \infty$. Then as $n \to \infty$, the distribution of

  $$\frac{\overline{Y} - \mu}{\sigma/\sqrt{n}}$$

  converges to a standard Normal distribution.

- Let $Y_1, Y_2, \ldots, Y_n$ be iid r.v.s with $\mathrm{E}[Y_i] = \mu_Y$ and $\mathrm{Var}[Y_i] = \sigma_Y^2 < \infty$. For large enough n,

  $$\frac{\overline{Y} - \mu_{\overline{Y}}}{\sigma_{\overline{Y}}} \quad \text{is approximately } N(0, 1^2).$$

- Let $Y_1, Y_2, \ldots, Y_n$ be iid r.v.s with $\mathrm{E}[Y_i] = \mu_Y$ and $\mathrm{Var}[Y_i] = \sigma_Y^2 < \infty$. Then as $n \to \infty$, the distribution of $\overline{Y}$ converges to a Normal distribution with mean μ_Y and standard deviation $\sigma_Y/\sqrt{n}$ (i.e., variance $\frac{\sigma_Y^2}{n}$).

Interpretation: For large enough n, the distribution of the sample mean is Normal; the variance of the sample mean decreases as n increases.

- How large is "large enough"?
  - If the $Y_i \sim N$ then $n = 1$ is enough
  - If the Y_i are very close to normal, e.g., a Uniform, then $n = 5$ or $n = 6$ may be enough
  - $n = 30 - 40$ is enough in most cases for continuous data (height, mpg, etc.)
  - We need a much larger sample size (1200 or more) if the Ys are Bernoulli r.v.s
- How fast does the variability decrease?
  - If you quadruple the sample size, then you cut the standard deviation of $\overline{Y}$ in half.

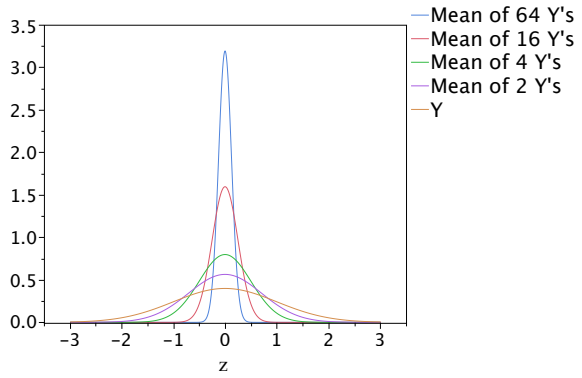
Cautions:

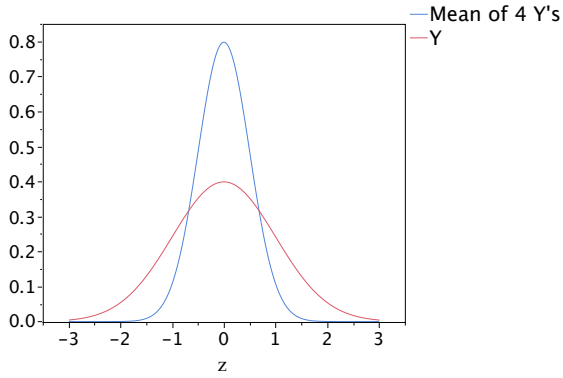
- The CLT does **NOT IMPLY** that the population distribution is Normal for large samples: the population distribution does not change!
- The CLT formulas use σ_Y^2, not S_Y^2. If you don't have a crystal ball, remember

$$\frac{\overline{Y}-\mu_Y}{S_Y/\sqrt{n}} \sim t_{n-1} \qquad \text{whenever} \qquad \frac{\overline{Y}-\mu_Y}{\sigma_Y/\sqrt{n}} \sim N(0,1^2)$$

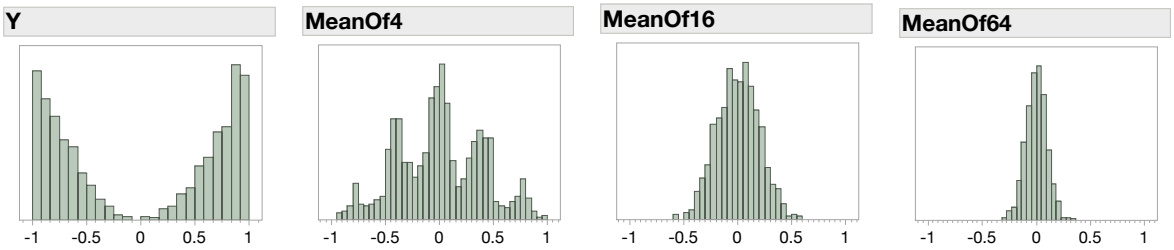
Pictures:

Here are some examples of the CLT in action. First up are several pictures where we look at what happens to a Normal random variate. On the left you see the distributions of sample means of different sizes when the original r.v. Y is a standard normal. As you can see, every time the sample size quadruples, the ± 3 std dev range is cut in half. This is a bit easier to see on the right: the pdf of Y drops close to zero at ± 3, the pdf of $\overline{Y}$ (based on $n = 4$) drops close to zero at ± 1.5.





You don't need to start with a Normal distribution for the CLT to apply. Here are some histograms of randomly-generated r.v.s, showing that even for a u-shaped distribution, when we take samples and compute the mean we end up with something that looks Normal.



B.3 Distributions for Modeling

There are several distributions commonly used in simulation modeling such as in the reference models of this book. We do not go into details on them all, but give a quick graphical overview.

Some of these distributions describe continuous-valued random variables.

- **Normal:** The Normal distribution, described earlier, is symmetric and bell-shaped. It is a good distribution for modeling phenomena that arise from aggregating many individual random effects, We remark that if a Normal is used to generate values that must be non-negative, such as service times, then a truncated Normal should be used unless the mean is quite far (in terms of standard deviations) above zero. A Normal random variable is typically parameterized in terms of its mean μ and standard deviation σ as Normal(μ, σ).

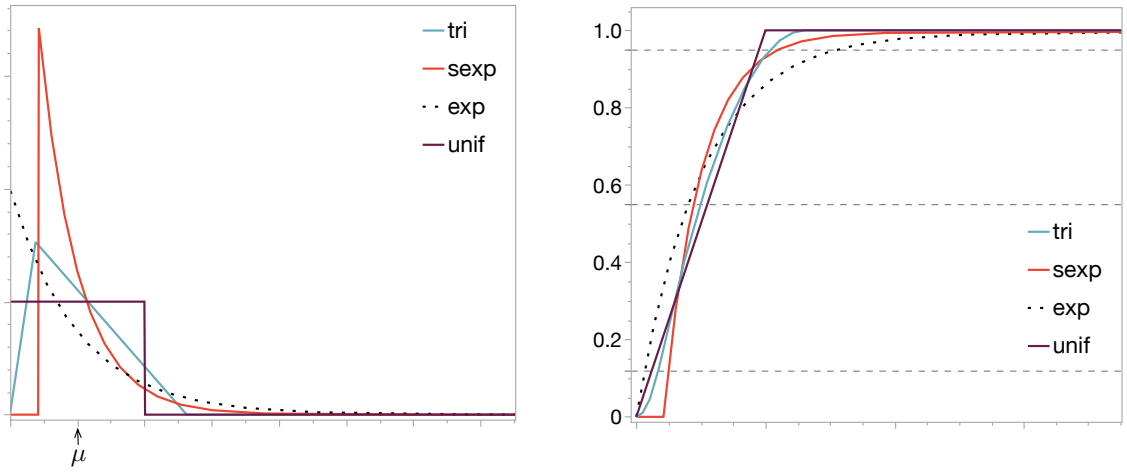
- **Uniform:** The uniform distribution is typically parameterized in terms of its minimum and maximum value. It is often used to model situations where we have little information, so the assumption that each value within the range is equally likely. Unif(a, b) has a mean of $(a + b)/2$ and a variance of $(b - a)^2/12$.

- **Exponential:** The exponential distribution has a single parameter, which can either be given as its mean μ or its rate λ, where $\lambda = 1/\mu$. When modeling make sure not to mix these up or your results may not make sense! Exponential distributions have some convenient mathematical properties, and are commonly used in stochastic modeling. In simulation they are often used to model the arrival process for a system to a system. Examples include the interarrival time between customer calls to a call center, or the operating time of a server before a breakdown occurs. The standard deviation is equal to the mean. Exponential(rate:λ) has a mean and standard deviation of $1/\lambda$.

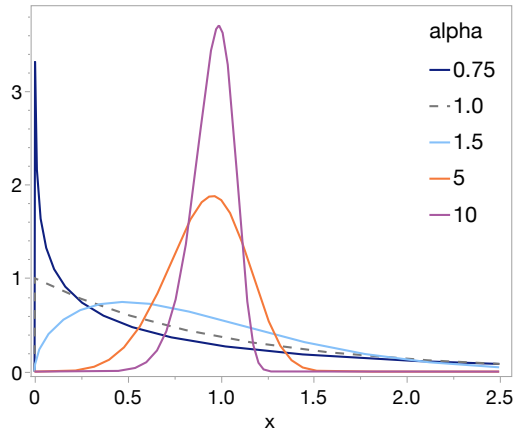
- **Shifted exponential:** One distribution we include in several models is a shifted exponential distribution. This shifts an exponential pdf to the right by some non-zero amount that represents the minimum possible value. We feel the shifted exponential is more appropriate for service time distributions than the exponential, because in our experience the most likely service time is not equal to zero.

- **Triangle:** The triangle distribution has three parameters: the minimum, the mode, and the maximum. It is often used as a simple surrogate, in part because it may be easier to get information about these three parameters from subject-matter experts. Examples might include service times (most likely to be 4 hours, but could be as little as 1 hour or as much as 10 hours). The triangle distribution can be symmetric, right-skewed, or left-skewed.

The pdfs and cdfs for representative members of the previous four distributions (uniform, exponential, shifted exponential, and triangle) appear below. Parameters were chosen so all distributions have the same means, and all except the exponential have the same standard deviations.



- **Weibull**: The Weibull distribution is often used to model time-to-failure in engineering applications because of its flexibility. It has two parameters: $\alpha > 0$ is the shape parameter and $\beta > 0$ is the scale parameter (note that other parameterizations are possible). The Weibull distribution is unimodal but can take on a variety of shapes, as shown below for various values of α when $\beta = 1$.



For a Weibull random variable, the mean and standard deviation of depend on both α and β. Specifically,

$$\mu = \Gamma\left(1 + \frac{1}{\alpha}\right) \qquad \text{and} \qquad \sigma^2 = \beta^2\left(\Gamma\left(1 + \frac{2}{\alpha}\right) - \Gamma\left(1 + \frac{1}{\alpha}\right)^2\right)$$

where Γ denotes the Gamma function (an extension of the factorial function for real numbers).

Other distributions are used to generate binary or discrete-valued random variates.

- **Bernoulli:** The Bernoulli distribution can be used to model binary outcomes, such as a coin flip or a true/false condition. If the two outcomes are coded as 0 or 1, then if $Pr(x = 1) = p$ a Bernoulli(p) distribution has mean p and variance $p(1 - p)$.
- **Poisson:** A Poisson distribution models the number of events occurring in a fixed interval of time or space, under the assumption that the events are independent and occur at a constant rate. It has a single parameter λ which is the mean number of events; λ is also the variance. The Poisson(λ) is often used in simulation as a model of an arrival process. It is related to the exponential distribution as follows: if the interarrival times are exponential, then the number of events in a fixed time is Poisson. A thinning method can be used to generate data from a non-homogeneous Poisson process (i.e., a process where the underlying rate changes over time) by generating potential events at the highest rate λ_{max} and then using a Bernoulli(λ/λ_{max}) to decide whether or not to retain a potential event schedule to occur when the rate is λ. Another type of application of a Poisson random variable in simulation occurs when modeling rare events, such as the number of defects in a semiconductor wafer.
- **Binomial:** The binomial distribution is another way of modeling random variables that are counts. The binomial has two parameters, n and p, and is the distribution that arises from summing n independent Bernoulli(p) random variables. In simulation modeling, this can be used to characterize counts such as the number of a fixed set of people who (independently) decide to purchase a product or get a vaccination on a particular day. A binomial(n, p) random variable has mean np and variance $np(1-p)$. If n is large the binomial can be approximated by a Normal distribution if $np > 10$ or a Poisson distribution with parameter $\lambda = np$ if $np \leq 10$.

Appendix C

Basic Manipulating Rules of Expectation and Variance

C.1 Notation

Throughout this review we will follow the common convention of using upper case letters such as X or Y to represent random variables, and lower case letters such as x or y to signify particular outcomes or values. We assume the reader is familiar with basic probability notation and manipulation at the level of bivariate distributions, conditional probability, and independence, for both discrete and continuous distributions. We also assume familiarity with standard mathematical symbology such as $\forall x$ ("for all x"), $A \iff B$ ("A if and only if B"), and summation and integration notation.

C.2 Random Variables

Recall that a random variable (RV) is a mapping of random events into the number line. For example, for a coin toss we can map the event that the outcome is tails to zero and the event that the outcome is heads to one. If the number of mappings is countable, i.e., can be placed into correspondence with the integers, we say that the RV is *discrete*; if uncountable, we say the RV is *continuous*. Note that having a finite number of outcomes implies countability, but being countable doesn't imply the outcomes are finite—distributions such as the Poisson or geometric are discrete but have an infinite number of possible outcomes.

When we are describing a discrete RV we refer to its *probability mass function* (pmf), which is a function that describes the probability that the random variable X has a particular outcome x as its value: $p_{\mathrm{X}}(x) \equiv P\{X = x\}$. Note that pmfs yield probabilities, so $0 \leq p_{\mathrm{X}}(x) \leq 1 \quad \forall x$

and

$$\sum_x p_{\mathrm{X}}(x) = 1. \tag{C.1}$$

For continuous random variables we can't talk about the probability of getting individual values. Instead, we use the *density function* $f_{\mathrm{X}}(x)$ to describe the relative likelihood of getting different values of x. Probabilities are associated with ranges, and are calculated as the area under the density function over the specified range:

$$P\{\ell \leq X \leq u\} = \int_\ell^u f_{\mathrm{X}}(x)dx. \tag{C.2}$$

As with the pmf, $f_{\mathrm{X}}(x) \geq 0 \quad \forall x$ and the total probability must be one:

$$\int_{-\infty}^{\infty} f_{\mathrm{X}}(x)dx = 1. \tag{C.3}$$

However, one of the quirks of calculus is that it is possible to have $f_{\mathrm{X}}(x) > 1$ as long as the area is one. If densities can be greater than one they are clearly *not* probabilities.

C.2.1 Exercises

Problem C.2.1. If X is a RV with pmf $p_{\mathrm{X}}(x) = k \cdot (5 - x)$ for $x = 1, \ldots, 4$, what is the value of k?

Problem C.2.2. Use Equation (C.2) to show why $P\{X = x\} = 0 \quad \forall x$ for continuous RVs.

Problem C.2.3. Let U be a RV which is uniformly distributed between a and b. Derive the density function for U.

Problem C.2.4. Let T be a RV which has a triangular distribution with the minimum and mode both 0 and the maximum 1.

a) Derive the density function for T.

b) Consider $f_{\mathrm{T}}(t)$ when $t = 0$. Why does this demonstrate that the density is not a probability?

C.3 Cumulative Distribution Functions

The *Cumulative Distribution Function* (CDF) for random variable X is defined to be

$$F_{\mathrm{X}}(x) \equiv P\{X \leq x\}. \tag{C.4}$$

CDFs are very useful because they give us a unified way of solving probability problems for ranges of random variables, regardless of whether the RVs are discrete or continuous.

Throughout the remainder of this discussion we will develop discrete and continuous cases in parallel, discrete on the left and continuous on the right. Equation (C.4) can therefore be calculated as

$$F_{\mathrm{X}}(x) = \sum_{u \le x} p_{\mathrm{X}}(u) \qquad \text{or} \qquad F_{\mathrm{X}}(x) = \int_{-\infty}^{x} f_{\mathrm{X}}(u)du. \tag{C.5}$$

CDFs have the following four properties:

1. the CDF yields a probability by its very definition, so $0 \le F_{\mathrm{X}}(x) \le 1 \quad \forall x$;
2. the CDF accumulates probability as we traverse the number line, so $F_{\mathrm{X}}(-\infty) = 0$ and $F_{\mathrm{X}}(\infty) = 1$;
3. $F_{\mathrm{X}}(x)$ is a monotonically non-decreasing function, i.e., $F_{\mathrm{X}}(x_1) \le F_{\mathrm{X}}(x_2) \quad \forall x_1 \le x_2$; and
4. $F_{\mathrm{X}}(x)$ is "right-continuous", i.e., at points of discontinuity the value of $F_{\mathrm{X}}(x)$ is found by taking the limit as you approach x from the right-hand side rather than from the left.

We can use the CDF of a given RV to calculate the probability that the outcome falls in some range, for example:

$$P\{\ell < X \le u\} = F_{\mathrm{X}}(u) - F_{\mathrm{X}}(\ell). \tag{C.6}$$

Note the strict inequality for the lower limit of the range—from the definition of a CDF we can see that the lower limit is not included as part of the range. In practice, we know that this has no impact for continuous RVs since $P\{X = x\} = 0 \ \forall x$, but it can radically affect our calculations for discrete RVs.

CDFs are often useful for generating random variates for use in computer simulations. RVs can often be generated from their CDFs based on the *inversion theorem*:

Theorem 1. *If the CDF of X is an invertible function then $F_X(X) = U$, where $U \sim$ Uniform(0,1).*

Inversion is accomplished by substituting the specific CDF of interest on the left-hand side of the theorem statement, then solving for X in terms of U. Any source of Uniform(0,1) random numbers can then be transformed to Xs with the desired distribution.

Our understanding of distributions extends straightforwardly to bivariate RVs. Let X and Y be random variables with joint CDF

$$F_{\mathrm{XY}}(x, y) \equiv P\{X \le x, Y \le y\}. \tag{C.7}$$

Similarly to the univariate case, this can be calculated as

$$F_{\mathrm{XY}}(x, y) = \sum_{u \le x} \sum_{v \le y} p_{\mathrm{XY}}(u, v) \qquad \text{or} \qquad F_{\mathrm{XY}}(x, y) = \int_{-\infty}^{x} \int_{-\infty}^{y} f_{\mathrm{XY}}(u, v)dvdu. \tag{C.8}$$

The marginal distributions of X and Y are found by summing or integrating across the joint distribution:

$$\begin{aligned} p_{\text{X}}(x) &= \sum_y p_{\text{XY}}(x,y) \\ p_{\text{Y}}(y) &= \sum_x p_{\text{XY}}(x,y) \end{aligned} \qquad \text{or} \qquad \begin{aligned} f_{\text{X}}(x) &= \int_{-\infty}^{\infty} f_{\text{XY}}(x,v)dv \\ f_{\text{Y}}(y) &= \int_{-\infty}^{\infty} f_{\text{XY}}(u,y)du \end{aligned} \tag{C.9}$$

$$X \text{ and } Y \text{ are independent} \iff \begin{cases} p_{\text{XY}}(x,y) = p_{\text{X}}(x)p_{\text{Y}}(y) & \forall x,y \qquad \text{(discrete RVs)} \\ f_{\text{XY}}(x,y) = f_{\text{X}}(x)f_{\text{Y}}(y) & \forall x,y \qquad \text{(continuous RVs).} \end{cases}$$

C.3.1 Exercises

Problem C.3.1. Derive the CDF of the uniform distribution from Problem C.2.3.

Problem C.3.2. Derive the CDF of the triangle distribution from Problem C.2.4.

Problem C.3.3. Derive the inversion for the uniform CDF from Problem C.3.1.

Problem C.3.4. Derive the inversion for the triangular CDF from Problem C.3.2.

Problem C.3.5. X and Y are continuous random variables with joint density function

$$f_{\text{XY}}(x,y) = \frac{3}{2}\left[x^2 - x^2y + y - 1\right] \qquad 1 \le x \le 2; \quad 0 \le y \le 1.$$

a) Derive the marginal density function for X.

b) Derive the marginal density function for Y.

c) Are X and Y independent?

d) Derive the joint CDF for X and Y.

e) Calculate $P\{X \le 1.25 \cap Y \le 0.4\}$.

C.4 Expectation

We will start by introducing the *law of the unconscious statistician* (*LotUS*), which tells us how to calculate the expected value of an arbitrary function of a random variable. This is a little unusual, but since all expectations can be calculated from it this will save time in the

long run. We will use a capital E to denote expected values. *LotUS* says that the expected value of function $g(X)$ can be calculated as

$$E[g(X)] = \begin{cases} \sum_x g(x) \cdot p_{\mathrm{X}}(x) & x \text{ discrete} \\ \\ \int_{-\infty}^{\infty} g(x) \cdot f_{\mathrm{X}}(x) dx & x \text{ continuous.} \end{cases} \tag{C.10}$$

For joint distributions this generalizes to

$$E[g(X,Y)] = \begin{cases} \sum_x \sum_y g(x,y) \cdot p_{\mathrm{XY}}(x,y) & x, y \text{ discrete} \\ \\ \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x,y) \cdot f_{\mathrm{XY}}(x,y) dy dx & x, y \text{ continuous.} \end{cases} \tag{C.11}$$

The mean of X, often denoted as μ_{X}, is the expected value of X. In other words, the function $g(X)$ is X itself:

$$\mu_{\mathrm{X}} = E[X] = \begin{cases} \sum_x x \cdot p_{\mathrm{X}}(x) & x \text{ discrete} \\ \\ \int_{-\infty}^{\infty} x \cdot f_{\mathrm{X}}(x) dx & x \text{ continuous.} \end{cases} \tag{C.12}$$

We can apply *LotUS* to quickly derive several important facts about expectations. The first of these is that for any constant c:

$$\begin{aligned} E[cX] &= \sum_x cx \cdot p_{\mathrm{X}}(x) & & E[cX] = \int_{-\infty}^{\infty} cx \cdot f_{\mathrm{X}}(x) \\ &= c\left(\sum_x x \cdot p_{\mathrm{X}}(x)\right) & \text{or} & = c\left(\int_{-\infty}^{\infty} x \cdot f_{\mathrm{X}}(x)\right) \\ &= cE[X] & & = cE[X]. \end{aligned} \tag{C.13}$$

Note that this result holds true regardless of whether the random variable is discrete or continuous. This is the case for the following two manipulation rules as well:

$$E[X+Y] = E[X] + E[Y] \qquad \text{always;} \tag{C.14}$$

$$X \text{ and } Y \text{ independent} \qquad \Longrightarrow \qquad E[XY] = E[X]E[Y]. \tag{C.15}$$

C.4.1 Exercises

Problem C.4.1. Show that Equation (C.14) is true for discrete distributions.

Problem C.4.2. Show that Equation (C.14) is true for continuous distributions.

Problem C.4.3. Show that Equation (C.15) is true for discrete distributions. Indicate where the independence assumption is required.

Problem C.4.4. Show that Equation (C.15) is true for continuous distributions. Indicate where the independence assumption is required.

Problem C.4.5. Let $X_1, \ldots, X_n$ be an identically distributed sample from a population with common mean μ_X, i.e., $E[X_i] = \mu_X$ for all i. Show that $\bar{X} = \sum_{i=1}^{n} X_i/n$ is an unbiased estimator of the common mean: $E[\bar{X}] = \mu_X$. Note that this does **not** require independence!

C.5 Variance, Covariance, and Correlation

The variance of random variable X is defined as:

$$\sigma_X^2 = Var(X) \equiv E\left[(X - \mu_X)^2\right]. \tag{C.16}$$

After expanding the square and applying our rules of expectation, this becomes

$$Var(X) = E\left[X^2\right] - \mu_X^2, \tag{C.17}$$

which is often computationally easier to calculate via *LotUS* than the definitional formula. Since we know from Equation (C.16) that variance is a quadratic form, it must *always* be a positive quantity.

A related measure is the *standard deviation* of the random variable. Standard deviation is found as the positive square root of variance: $\sigma_X = \sqrt{\sigma_X^2}$. The standard deviation has the same units as X and is generally thought of as a yardstick for measuring how far away individual values are from the center of mass of the distribution.

If we are dealing with two random variables we can see whether they have a tendency to be linearly related using *covariance*. The covariance between X and Y is defined as

$$\gamma_{XY}^2 = Cov(X, Y) \equiv E\left[(X - \mu_X) \cdot (Y - \mu_Y)\right]. \tag{C.18}$$

As with variance, we can expand the product and apply our rules of expectation to yield

$$Cov(X, Y) = E\left[XY\right] - \mu_X \mu_Y. \tag{C.19}$$

Unlike variance, in general covariance can be either a positive or a negative quantity. If X and Y tend to be linearly related, the sign of the covariance will indicate whether the slope

of the linear relationship is positive or negative. As an aside, note that variance is actually a special case of covariance, i.e., σ_{X}^2 can be expressed as $Cov(X, X)$.

Note that if X and Y are independent, $Cov(X, Y) = 0$. However, the converse is not always true—having a covariance of zero does not guarantee independence.

One drawback to using covariance is that numerical results depend on the units. For instance, if Y is the stopping distance of a vehicle and X is the initial velocity, we get entirely different numeric answers depending on whether we measure Y in inches, meters, or miles and X in meters per second, miles per hour, or furlongs per fortnight. We eliminate this units dependence by scaling covariance by the standard deviations of X and Y. The result is known as the *correlation* between X and Y, and is denoted using the Greek letter rho:

$$\rho_{\text{XY}} = \frac{\gamma_{\text{XY}}^2}{\sigma_{\text{X}}\sigma_{\text{Y}}}. \tag{C.20}$$

Correlation is a unitless measure. We will find the same correlation between initial velocity and stopping distance regardless of what units we use to measure each of them. If X and Y are independent, then their correlation will be zero because the numerator is zero. A triangle inequality can be used to prove that $|\rho_{\text{XY}}| \leq 1$. Since the denominator is positive, the sign of correlation is determined by the sign of its numerator (covariance). The magnitude can be interpreted as a direct measure of the degree of linear relationship between X and Y, with zero meaning there is no linear relationship and ± 1 indicating a perfect linear relationship.

Pulling all the pieces together, we can derive the following important relationships for variances:

$$Var(X + c) = Var(X); \tag{C.21}$$

$$Var(cX) = c^2 Var(X); \tag{C.22}$$

$$\text{and} \quad Var(X \pm Y) = Var(X) + Var(Y) \pm 2Cov(X, Y), \tag{C.23}$$

where the $\pm$ signs must be in agreement on the left and right sides of the equality. Note that if X and Y are independent, then Equation (C.23) reduces to

$$Var(X \pm Y) = Var(X) + Var(Y). \tag{C.24}$$

C.5.1 Exercises

Problem C.5.1. Use the properties of expectation established in Section C.4 to derive Equation (C.17) from Equation (C.16).

Problem C.5.2. Use the properties of expectation established in Section C.4 to derive Equation (C.19) from Equation (C.18).

Problem C.5.3. Use the properties of expectation established in Section C.4 and Equation (C.19) to show that $Cov(X, Y) = 0$ when X and Y are independent.

Problem C.5.4. Let X be a RV which is distributed uniformly between -1 and 1, and let $Y = X^2$.

a) Are X and Y independent of each other? Why or why not?

b) Use Equation (C.19) and *LotUS* to calculate γ^2_{XY}. Discuss your result in light of your answer to part (a).

Problem C.5.5. Show that Equation (C.21) is true.

Problem C.5.6. Show that Equation (C.22) is true.

Problem C.5.7. Show that Equation (C.23) is true.

Appendix D

Computational Complexity

Pronunciation: The Greek letter Θ (theta) is pronounced "THAY-tah" and the Greek letter Ω (omega) is pronounced "oh-MAY-gah."

Three notations are used to express computational complexity of a task. They are identified as O (often referred to as big-O), Ω, and Θ.

Big-O notation describes an upper bound on the rate of growth for the time, number of operations involved, or memory or storage resources required by an algorithm as a function of the number of elements being processed. For instance, we may be interested in how $T(n)$, the amount of time a sorting algorithm requires, varies as a function of n, the number of elements to be sorted. We say that $T(n) \in \mathrm{O}(f(n))$ if

$$\exists \quad k > 0, \quad n' > 0 \qquad s.t. \quad T(n) \leq k \cdot f(n) \qquad \forall n \geq n'.$$

This is often written or stated in shortened form to $T(n) = \mathrm{O}(f(n))$ or "$T(n)$ is $\mathrm{O}(f(n))$," respectively. As an example of Big-O, many comparison-based sorting algorithms such as bubble sort or insertion sort are $\mathrm{O}(n^2)$. Please note that this describes how problems scale, it does *not* give specific predictions of run-times or resource utilization or tell you which algorithm will be faster in a particular context. If an algorithm is $\mathrm{O}(n^2)$, doubling the number of elements will quadruple the run time, while increasing the number by an order of magnitude will increase the run time by two orders of magnitude. However, it is entirely possible than an $\mathrm{O}(n^2)$ algorithm may be faster than an $\mathrm{O}(n \log n)$ algorithm over the intended range of usage due to the magnitude of the scaling constants involved.

Similarly, Ω notation describes a lower bound on rates of growth. We say that $T(n) \in \Omega(f(n))$ if

$$\exists \quad k > 0, \quad n' > 0 \qquad s.t. \quad T(n) \geq k \cdot f(n) \qquad \forall n \geq n'.$$

For example, computer scientists have proven that comparison-based sorting algorithms are in $\Omega(n \log n)$ when the data are not sorted *a priori*. Variants of the quicksort algorithm all

have best-case behavior that is proportional to $n \log n$, but worst case behavior proportional to n^2, so quicksort is in both $\Omega(n \log n)$ and $\mathrm{O}(n^2)$.

When an algorithm has both $T(n) \in \mathrm{O}(g(n))$ and $T(n) \in \Omega(g(n))$ for the same function $g(n)$, we can say that $T(n) \in \Theta(g(n))$. In other words, $g(n)$ represents the actual rate of growth for the algorithm. For example, we *cannot* say that quicksort is in $\Theta(n \log n)$ or $\Theta(n^2)$ since the upper and lower bounds on its performance differ. However, merge sort and heap sort are both in $\Theta(n \log n)$ because both their worst-case and best-case performances are proportional to $n \log n$.

Appendix E

Efficient Dynamic Updating for Descriptive Statistics

Kalman filtering is a parameter estimation technique that specifies how to update an already existing estimate when one additional observation is added. We will derive a Kalman filter-like estimator for sample means to illustrate how it's done, and provide a corresponding estimator for sample variance without showing the derivation.

Most statistics texts talk about how to estimate the sample mean $\bar{X}$ after all of the observations have become available. In practice the observations usually arrive sequentially. This means that we actually have a sequence of sample means: $\bar{X}_1, \bar{X}_2, \bar{X}_3, \ldots, \bar{X}_k, \ldots, \bar{X}_n$, where

$$\bar{X}_k \equiv \begin{cases} \text{undefined} & \text{for } k \leq 0; \\ \\ \dfrac{1}{k}\displaystyle\sum_{i=1}^{k} X_i & \text{for } k > 0. \end{cases}$$

The subscript k denotes the number of items in the sample upon which the average is based. We can define an update operation using recurrence:

$$\bar{X}_k = \begin{cases} X_1 & k = 1; \\ \\ \bar{X}_{k-1} + \Delta_{\bar{X}} & k > 1. \end{cases}$$

Our objective is to determine the value of $\Delta_{\bar{X}}$. We can do this using algebra after rearranging

the recurrence for $k > 1$:

$$\begin{aligned}
\Delta_{\bar{X}} &= \bar{X}_k - \bar{X}_{k-1} \\
\Delta_{\bar{X}} &= \frac{1}{k}\sum_{i=1}^{k} X_i - \frac{1}{k-1}\sum_{i=1}^{k-1} X_i \\
k(k-1)\Delta_{\bar{X}} &= (k-1)\sum_{i=1}^{k} X_i - k\sum_{i=1}^{k-1} X_i \\
k(k-1)\Delta_{\bar{X}} &= (k-1)\left[\sum_{i=1}^{k-1} X_i + X_k\right] - k\sum_{i=1}^{k-1} X_i \\
k(k-1)\Delta_{\bar{X}} &= (k-1)X_k - \sum_{i=1}^{k-1} X_i \\
k\Delta_{\bar{X}} &= X_k - \frac{1}{k-1}\sum_{i=1}^{k-1} X_i \\
k\Delta_{\bar{X}} &= X_k - \bar{X}_{k-1} \\
\Delta_{\bar{X}} &= \frac{1}{k}\left(X_k - \bar{X}_{k-1}\right).
\end{aligned}$$

If we define $\delta_k \equiv \left(X_k - \bar{X}_{k-1}\right)$, then $\Delta_{\bar{X}} = \delta_k/k$ and our update recurrence becomes

$$\bar{X}_k = \begin{cases} X_1 & k = 1; \\ \bar{X}_{k-1} + \dfrac{\delta_k}{k} & k > 1. \end{cases}$$

Note that as long as all the observations are finite, $\Delta_{\bar{X}} \to 0$ as $k \to \infty$. This makes it clear that in the limit $\bar{X}_k$ converges to a constant, usually designated μ.

A similar result can be derived for

$$s_k^2 \equiv \begin{cases} \text{undefined} & \text{for } k \leq 1; \\ \dfrac{1}{k-1}\displaystyle\sum_{i=1}^{k}(X_i - \bar{X}_k)^2 & \text{for } k > 1. \end{cases}$$

the variance estimator based on k observations. The derivation been left as an exercise for the interested student, but when the dust settles the recurrence relationship is:

$$s_k^2 = \begin{cases} \text{undefined} & \text{for } k \leq 1; \\ \left(\dfrac{k-2}{k-1}\right) s_{k-1}^2 + \dfrac{\delta_k^2}{k} & k > 1, \end{cases}$$

where δ_k is the same value used to update $\bar{X}_k$. As with the sample mean update, $\delta_k^2/k \to 0$ as $k \to \infty$ if all the observations are finite, while at the same time $(k-2)/(k-1) \to 1$. Thus s_k^2 also converges to a constant, usually designated σ^2.

An alternative which avoids repeatedly updating the $(k-2)/(k-1)$ rescaling is to work with the sum of squared deviations (SSD), which has the following recurrence relationship:

$$SSD_k = \begin{cases} \text{undefined} & \text{for } k \leq 1; \\ \\ SSD_{k-1} + \delta_k \cdot (X_k - \bar{X}_k) & k > 1. \end{cases}$$

The sample variance can then be calculated at any point as $SSD_k/(k-1)$.

Bibliography

Ankenman, Bruce, Barry L. Nelson, Jeremy Staum. 2008. Stochastic kriging for simulation metamodeling. S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler, eds., *Proceedings of the 2008 Winter Simulation Conference*. Institute of Electrical and Electronic Engineers, Inc., Piscataway, New Jersey, 362–370. URL `https://www.informs-sim.org/wsc08papers/042.pdf`.

Ankenman, Bruce, Barry L. Nelson, Jeremy Staum. 2010. Stochastic kriging for simulation metamodeling. *Operations Research* **58**(2) 362–370.

Binois, Mickael, Robert B. Gramacy, Michael Ludkovski. 2016. Practical heteroskedastic Gaussian process modeling for large simulation experiments.

Box, G. E. P. 1988. Signal-to-noise ratios, performance criteria, and transformations (with discussions). *Technometrics* **30**(1) 1–40.

Cioppa, Thomas M., Thomas W. Lucas. 2007. Efficient nearly orthogonal and space-filling Latin hypercubes. *Technometrics* **49**(1) 45–55.

Duan, Weitao, Bruce E Ankenman, Susan M Sanchez, Paul J Sanchez. 2017. Sliced full factorial-based latin hypercube designs as a framework for a batch sequential design algorithm. *Technometrics* **59**(1) 11–22.

Elmegreen, Bruce E., Susan M. Sanchez, Alex S. Szalay. 2014. The future of computerized decision making. A. Tolk, S. Y. Diallo, O. Ryzhov, L. Yilmaz, S. Buckley, J. A. Miller, eds., *Proceedings of the 2014 Winter Simulation Conference*. Institute of Electrical and Electronic Engineers, Inc., Piscataway, New Jersey, 943–949.

Erickson, Collin, Bruce E. Ankenman, Susan M. Sanchez. 2018. Comparison of gaussian process modeling software. *European Journal of Operational Research* **266**(1) 179–192.

Ewald, R., A. M. Uhrmacher. 2014. SESSL: A domain-specific language for simulation experiments. *ACM Transactions on Modeling and Computer Simulation* **24**(2) 11:1–11:25.

Feldkamp, Niclas, Soeren Bergmann, Steffen Strassburger. 2020a. Knowledge discovery in simulation data. *ACM Transactions on Modeling and Computer Simulation,* forthcoming forthcoming.

Feldkamp, Niclas, Soeren Bergmann, Steffen Strassburger. 2020b. Knowledge discovery in simulation data. *ACM Transactions on Modeling and Computer Simulation,* forthcoming **30**(4) Article 24. 1–25.

Garber, M. 2013. Computing power used to be measured in 'kilo-girls'. *The Atlantic* **October 16**. URL `https://www.theatlantic.com/technology/archive/2013/10/computing-power-used-to-be-measured-in-kilo-girls/280633/`.

Hernandez, Alejandro S., Thomas W. Lucas, Matthew Carlyle. 2012. Enabling nearly orthogonal Latin hypercube construction for any non-saturated run-variable combination. *ACM Transactions on Modeling and Computer Simulation* **22**(4) 20:1–20:17.

Hsu, J. 1981. Simultaneous confidence intervals for all distances from the 'best'. *Annals of Statistics* **9**(1026–1034).

HTCondor.org. 2025. Htcondor software. online. URL `https://htcondor.org/`.

Hunker, J. 2024. Method for combining data farming and data mining in a logistics assistance system for materials trading networks based on graph databases. Ph.D. thesis, Technical University Dortmund, Dortmund, Germay.

Kalman, R. E. 1963. Mathematical description of linear dynamical systems. *Journal of the Society for Industrial and Applied Mathematics, Series A: Control* **1**(2) 152–192.

Kleijnen, J. P. C. 2015. *Design and Analysis of Simulation Experiments.* 2nd ed. Springer, New York.

Kleijnen, J. P. C., S. M. Sanchez, T. W. Lucas, T. M. Cioppa. 2005. A user's guide to the brave new world of designing simulation experiments. *INFORMS Journal on Computing* **17**(3) 263–289.

Li, Hung-xin. 2015. Improving the Taiwan military's disaster relief response to typhoons. Master's thesis, Naval Postgraduate School, Monterey, CA, Monterey, CA. URL `https://hdl.handle.net/10945/45891`.

Lucas, Thomas W., W. David Kelton, P. J. Sánchez, S. M. Sanchez, Ben L. Anderson. 2015. Changing the paradigm: Simulation, now a method of first resort. *Naval Research Logistics* **62**(4) 293–303.

MacCalman, A. D., H. Vieira, T. W. Lucas. 2017. Second-order nearly orthogonal latin hypercubes for exploring stochastic simulations. *Journal of Simulation* **11**(2) 137–150.

MacCalman, Alex D., Susan M. Sanchez, Mary L. McDonald, Simon R. Goerger, Andrew T. Karl. 2016. Tradespace analysis for multiple performance measures. Theresa M. K. Roeder, Peter I. Frazier, Roberto Szechtman, Enlu Zhou, Todd Huschka, Stephen E. Chick, eds., *Proceedings of the 2016 Winter Simulation Conference.* IEEE, Piscataway, NJ, 3063–3074. URL `https://www.informs-sim.org/wsc16papers/270.pdf`.

Marlow, David, S. M. Sanchez, Paul J. Sanchez. 2015. Testing aircraft fleet management policies using simulation experimental design. *MODSIM2015, 21th International Congress on Modelling and Simulation*. Modelling and Simulation Society of Australia and New Zealand, 917–923.

Marlow, David O., Susan M. Sanchez, Paul J. Sanchez. 2019. Testing policies and key influences on long-term aircraft fleet management using designed simulation experiments. *Military Operations Research* **24**(3) 5–25.

Matković, Krešimir, Denis Gračanin, Helwig Hauser. 2018. Visual analytics for simulation ensembles. Marcus Rabe, Angel A. Juan, Navonil Mustafee, Anders Skoogh, Sanjay Jain, Björn Johansson, eds., *Proceedings of the 2018 Winter Simulation Conference*. IEEE, Piscataway, NJ, 321–335.

Meketon, Marc S., Bruce Schmeiser. 1984. Overlapping batch means: Something for nothing? S. Sheppard, U. Pooch, D. Pegden, eds., *Proceedings of the 1984 Winter Simulation Conference*. Institute of Electrical and Electronics Engineers, Inc., Piscataway, NJ, 227–230.

Mitroff, I. I., T. R. Featheringham. 1974. On systemic problem solving and the error of the third kind. *Behavioral Science* **19**(6) 383–393.

Moen, R. 2020. Foundation and history of the pdsa cycle. URL `https://deming.org/wp-content/uploads/2020/06/PDSA_History_Ron_Moen.pdf`.

Morgan, B. L., H. C. Schramm, J. R. Smith, T. W. Lucas, M. L. McDonald, P. J. Sanchez, S. M. Sanchez, S. C. Upton. 2018. Improving u.s. navy campaign analysis using big data. *Interfaces* **48**(2) 130–146.

Nair, Vijayan N., Bovas Abraham, Jock MacKay, George Box, Raghu N. Kacker, Thomas J. Lorenzen, James M. Lucas, Raymond H. Myers, G. Geoffrey Vining, John A. Nelder, Madhav S. Phadke, Jerome Sacks, William J. Welch, Anne C. Shoemaker, Kwok L. Tsui, Shin Taguchi, C. F. Jeff Wu. 1992. Taguchi's parameter design: A panel discussion. *Technometrics* **34**(2) 127–161. doi:10.1080/00401706.1992.10484904. URL `https://amstat.tandfonline.com/doi/abs/10.1080/00401706.1992.10484904`.

NATO. 2014. Data farming in support of NATO. Tech. Rep. TR-MSG-088, NATO Science & Technology Organization Technical Report. URL `https://www.cso.nato.int/Pubs/rdp.asp?RDP=STO-TR-MSG-088`.

Nelson, B. L. 2022. Let's do ranking & selection. B. Feng, J. Pedrielli, Y. Peng, S. Shashaani, E. Song, C. G. Corlu, L.H Lee, E. P. Dhew, T. Roeder, P. Lendermann, eds., *Proceedings of the 2022 Winter Simulation Conference*. IEEE, Piscataway, NJ, 60–74.

Nelson, B. L., J. Swann, D. Goldsman, W. Song. 2001. Simple procedures for selecting the best simulated system when the number of alternatives is large. *Operations Research* **49**(6) 950–963.

Pignatiello, J. J. Jr., J.S. Ramberg. 1991. Top ten triumphs and tragedies of Genichi Taguchi. *Quality Engineering* **4**(2) 211–235.

Ramberg, J. S., J. J. Jr. Pignatiello, S. M. Sanchez. 1992. A critique and enhancement of the Taguchi method. *ASQC Quality Congress Transactions* 491–498.

Ramberg, J. S., S. M. Sanchez, P. J. Sanchez, L. J. Hollick. 1991. Designing simulation experiments: Taguchi methods and response surface metamodels. B. L. Nelson, W. D. Kelton, G. M. Clark, eds., *Proceedings of the 1991 Winter Simulation Conference*. Institute of Electrical and Electronics Engineers, Inc., Piscataway, NJ, 167–176.

Rasmussen, Carl Edward, Christopher K. I. Williams. 2006. Gaussian processes for machine learning. *The MIT Press* **abs/1309.6835**. URL `http://www.gaussianprocess.org/gpml/chapters/RW.pdf`.

Reed, L. B. 1938. Advantages of statistical methods for the practical entomologist. *The Florida Entomologist* **XXI**(3) 33–38.

Sacks, J., W. J. Welch, T. J. Mitchell, H. P. Wynn. 1989. Design and analysis of computer experiments (includes comments and rejoinder). *Statistical Science* **4** 409–435.

Sanchez, P. J. 2009. Fundamentals of simulation modeling. *Engineering Management Review* **37**(2) 23–33.

Sanchez, Paul J., Susan M. Sanchez. 2019. Orthogonal second-order space-filling designs with insights from simulation experiments to support test planning. *Quality and Reliability Engineering International* **35**(3) 854–867.

Sanchez, Paul J., K. P. White. 2011. Interval estimation using replication/deletion and mser truncation. S. Jain, R.R. Creasey, J. Himmelspach, K.P. White, M. Fu, eds., *Proceedings of the 2011 Winter Simulation Conference*. IEEE, 488–494.

Sanchez, S. M. 2021. Data farming: the meanings and methods behind the metaphor. M. Fakimi, D. Robertson, T. Boness, eds., *Proceedings of the Operational Research Society Simulation Workshop 21 (SW21)*. 10–17. doi:10.36819/SW21.002. URL `https://doi.org/10.36819/SW21.002`.

Sanchez, S. M., T. W. Lucas. 2002. Exploring the world of agent-based simulation: Simple models, complex analyses. E. Yucësan, C.-H. Chen, J. L. Snowdon, J. Charnes, eds., *Proceedings of the 2002 Winter Simulation Conference*. Institute of Electrical and Electronics Engineers, Inc., Piscataway, New Jersey, 116–126.

Sanchez, S. M., P. J. Sanchez. 2005. Very large fractional factorial and central composite designs. *ACM Transactions on Modeling and Computer Simulation* **15**(4) 362–377.

Sanchez, S. M., P. J. Sanchez. 2017. Better big data via data farming experiments. A. Tolk, J. Fowler, G. Shao, E. Yücesan, eds., *Advances in Modeling and Simulation: Seminal Research from 50 Years of Winter Simulation Conferences*. Springer International Publishing, Cham, Switzerland, 159–179.

Sanchez, S. M., P. J. Sanchez, J. S. Ramberg. 1998. A simulation framework for robust system design. B. Wang, ed., *Concurrent Design of Products, Manufacturing Processes and Systems*, chap. 12. Gordon and Breach, New York, 279–314.

Sanchez, S. M., L. D. Smith, E. C. Lawrence. 1996. Sensitivity and scenario analysis for simulation metamodels. J. M. Charnes, D. J. Morrice, T. Brunner, J. J. Swain, eds., *Proceedings of the 1996 Winter Simulation Conference*. Institute of Electrical and Electronic Engineers, Piscataway, New Jersey, 1440–1447.

Sanchez, Susan M. 2018. Data farming: better data, not just big data. Marcus Rabe, Angel A. Juan, Navonil Mustafee, Anders Skoogh, Sanjay Jain, Björn Johansson, eds., *Proceedings of the 2018 Winter Simulation Conference*. IEEE, Piscataway, NJ, 425–439.

Sanchez, Susan M. 2020. Data Farming: Methods for the Present, Opportunities for the Future. *ACM Transactions on Modeling and Computer Simulation* **30**(4) Article 22. 1–30.

Sanchez, Susan M., Paul J. Sanchez. 2020. Robustness revisited: Simulation optimization viewed through a different lens. Ki-Hwan G. Bae, Ben Feng, Sojung Kim, Sanja Lazarova-Molnar, Zeyu Zheng, Theresa Roeder, Renee Thiesing, eds., *Proceedings of the 2020 Winter Simulation Conference*. IEEE, Piscataway, NJ, 60–74.

Santner, T. J., B. J. Williams, W. I. Notz. 2003. *The Design and Analysis of Computer Experiments*. Springer-Verlag, New York.

Scheaffer, R. L., A. Watkins, M. Gnanadesikan, J. A. Witmer. 1996. *Activity-Based Statistics*. Springer-Verlag, New York.

Schruben, L. W., S. M. Sanchez, P. J. Sanchez, V. A. Czitrom. 1992. Variance reallocation in Taguchi's robust design framework. *Proceedings of the 2009 Winter Simulation Conference*. 548–556.

Schruben, Lee W. 2017. Model is a verb. Andreas Tolk, John Fowler, Guodong Shao, Enver Yücesan, eds., *Advances in Modeling and Simulation*. Springer International Publishing AG, Cham, Switzerland, 17–26.

Smith, J. S., D. T. Sturrock. 2024. *Simio and Simulation: Modeling, Analysis, Applications*. 7th ed. Simio. URL `https://textbook.simio.com/SASMAA7/`.

Sobol, Ilya Meerovich. 1967. On the distribution of points in a cube and the approximate evaluation of integrals. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki* **7**(4) 784–802.

Taguchi, G. 1987. *System of Experimental Design*, vol. 1 and 2. UNIPUB/Krauss International, White Plains, NY.

Taguchi, G., Y. Wu. 1980. *Introduction to Off-line Quality Control*. Central Japan Quality Association, Nagoya, Japan.

Tukey, J. W. 1962. The future of data analysis. *The Annals of Mathematical Statistics* **33**(1) 1–67. URL `http://www.jstor.org/stable/2237638`.

Vieira Jr, Hélcio, S. M. Sanchez, K. H. K. Kienitz, M. C. N. Belderrain. 2013. Efficient, nearly orthogonal-and-balanced, mixed designs: An effective way to conduct trade-off analyses via simulation. *Journal of Simulation* **7**(4) 264–275.

Vieira Jr, Hélcio, Susan M. Sanchez, M. McDonald. 2022. Noab_mixeddesigns.xlsx spreadsheet. URL `https://harvest.nps.edu`.

Warnke, T., A. Uhrmacher. 2018. Complex simulation experiments made easy. M. Rabe et al., ed., *Proceedings of the 2018 Winter Simulation Conference*. IEEE, Piscataway, New Jersey, 410–424.

Welch, W. J., T. K. Yu, S. M. Kang, J. Sacks. 1990. Computer experiments for quality control by robust design. *Journal of Quality Technology* **22** 15–22.

Wharton, D. I., F. K. C. Hui. 2011. The arcsine is asinine: the analysis of proportions in ecology. *Ecology* **92**(1) 3–10.

Wolf, Eric. 2003. Using agent-based distillations to explore logistics support to urban, humanitarian assistance/disaster relief operations. Master's thesis, Naval Postgraduate School.