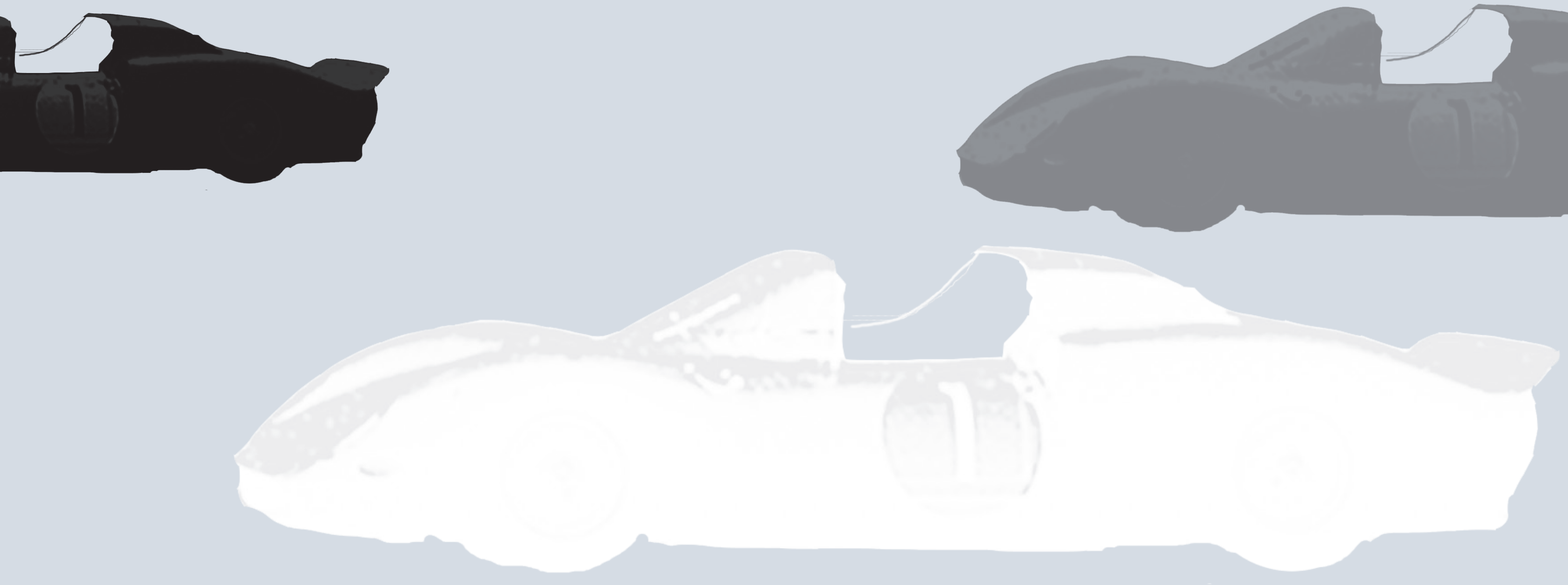
TECHNOLOGY PARTNERZ

The Need for Speed 2019

BY
ERIC TORKIA, MASC & ROBERT D. BROWN III



A special thanks to Dr. Sam Savage, Dr. William McKibbin, Michael Albo of the Decision Science Institute and Eric Wainwright who contributed their time and insights to make this a better paper.

THE NEED FOR SPEED 2019

BENCHMARKING SIMULATION FOR SPEED AND ACCURACY

Tests conducted using a Markowitz Mean Variance Optimization Model

For more information on obtaining reproduction rights or customized educational white papers :

TECHNOLOGY PARTNERZ LTD.

50 St-Charles St. W., Suite 100
Longueuil, Qc., Canada
J4H 1C6

+888-879-8440 (Toll-Free)
+514-278-2221 (Montreal)

info@technologypartnerz.com

Contents

INTRODUCTION

In this article, we are going to cover aspects of Monte Carlo tools which we believe will interest many readers – speed, accuracy and precision. We look at both dimensions of speed, development speed and numerical processing speed. Speeding up an analytics cycle-time to an answer is both about the time to model and the time to insight.

Picking the candidates

The official release of the Julia Language which many consider to be the up and coming language to rival both R and Python, prompted this analysis. We have been keeping a close watch on Julia since 2014 because of the promises of ease and speed. Given that the language was still in beta we wanted to see if it would take. Now we are confident enough to have spent weeks going through the process of learning and benchmarking Julia against R, @RISK and Oracle Crystal Ball. We had considered Python because we feel it's a solid platform with lots of packages and can perform well when combined with Numba, however we were not a able to get this contribution in before publishing but we will revisit in our next study.

In the 8 years that have passed we have seen many things change and others stay the same in the risk and decision modeling software space. Though @RISK and Crystal ball have had many releases since our first NFS (Need for Speed) comparison study, only in recent years has 64bit versions of all the tools tested become available. This time around we only tested correlated models (assuming that anything less is not useful) and focused on simulating returns using Normal Distributions in 64bit environments. Of course, we could have also tested distribution fitting but that could distort the results, so we left that out also. What we did focus on what the consistency of results and the time it takes to build a reliable solution.

Comparing Excel and Code approaches to simulation

It goes without saying that coding a simulation model and developing one in Excel with advanced add-ins that both automate and simplify the Monte-Carlo modeling process are indeed two very different undertakings. Basically it's a trade-off between development time and the time it take to update the inputs and run again (automation into the business process). Coding a solution can take 10-15x more time to develop than it's equivalent Excel counterpart but deliver run times and the ability update and refresh inputs automatically making the coded solution 100x – 10000X faster on the back end.

Given that many of our clients are seeking to leverage/share existing models and put them in broad use, we need to identify a language that was practical, fast, intuitive to code and easy to read. Another core consideration is that it needed to be a natural evolution for someone who is proficient in Microsoft Excel and VBA because again, we often get the question : "What language should I learn if I know Excel VBA?". So in this review, we consider both Julia and R as counters to Excel.

Methodology for testing the software

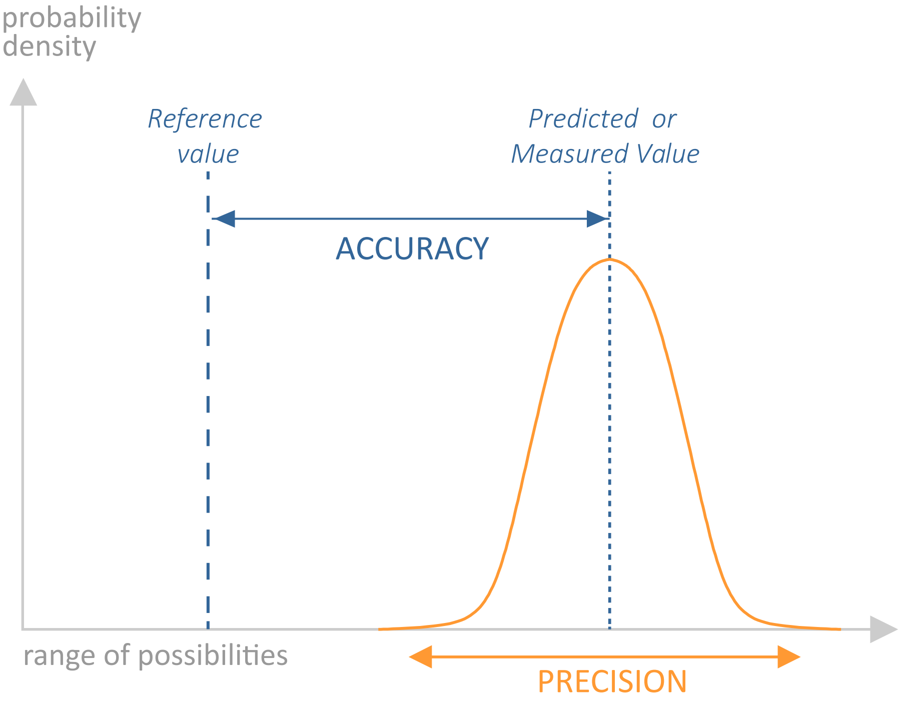
Our curiosity was piqued when we ran several simulations and noticed that the standard deviation was moving around at certain percentiles at different rates than others. Therefore, we started wondering how much volatility would naturally occur at various percentiles across different packages.

Our testing focused on two different aspects: *accuracy and precision*. We used the correlated returns model from our previous article as our test model. We ran the model 20 times at 10 000, 50 000 and 100 000 trials, resulting in 60 simulations per package.

Testing for accuracy & precision

In order to test for accuracy we averaged out the values of all 20 simulations for each percentile/package and compared the results against the calculated form using the Markowitz mean variance approach. As for precision, we looked at the error (theoretical *minus* simulated results) of each percentile/package at a given number of trials for all 20 simulations. For example, we would look at the standard deviation at the 99th percentile using a sample of 20 simulations at 10,000 trials each (200,000 trials total).

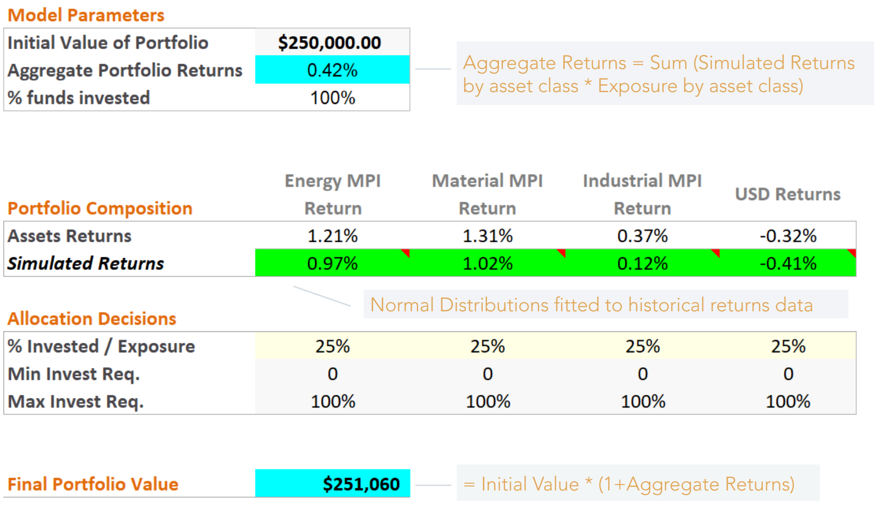
Our basic objective when analyzing a package for precision is whether it follows a certain set of statistical rules. A good example would be the expectation for a reduction of standard deviation as we increase the number of trials at a given percentile.



When we set up the script to test the accuracy of each tool we realized that it would be easy to do a performance comparison as well. This turned out to be an added bonus! During the scripted tests, we also took into account the execution time of the whole script and that of each simulation. We also evaluated the average trials/sec for both the test and the individual simulations.

Test model for benchmarking development and run times

An investor seeks to evaluate the risk in his portfolio. We have $250,000 to invest and must make sure we have a better chance at making money than losing it. For purposes of the exercise, we are using the Markowitz Modern Portfolio Theory (1952) and allocated our investment equally among each asset class at 25% a piece. We analyzed the rank correlation (using the Excel based method presented Copulas vs. Correlation) and prepared a correlation matrix that was used to correlate the returns distributions for each class. In Julia, we replicated all of the logic from Excel and it's add-ins, including the use of the Iman-Conover (1982) correlation algorithm to correlate the assets together.



Model Parameters

Initial Value of Portfolio	$250,000.00
Aggregate Portfolio Returns	0.42%
% funds invested	100%

Aggregate Returns = Sum (Simulated Returns by asset class * Exposure by asset class)

Portfolio Composition	Energy MPI Return	Material MPI Return	Industrial MPI Return	USD Returns
Assets Returns	1.21%	1.31%	0.37%	-0.32%
Simulated Returns	0.97%	1.02%	0.12%	-0.41%

Normal Distributions fitted to historical returns data

Allocation Decisions				
% Invested / Exposure	25%	25%	25%	25%
Min Invest Req.	0	0	0	0
Max Invest Req.	100%	100%	100%	100%

Final Portfolio Value	$251,060

= Initial Value * (1+Aggregate Returns)

Correlation Matrix used by the portfolio model

	Energy MPI Return	Material MPI Return	Industrial MPI Return	USD Returns
Energy Returns	1.0000	0.4740	0.3271	0.0026
Material Returns	0.4740	1.0000	0.3105	-0.0270
Industrial Returns	0.3271	0.3105	1.0000	-0.0554
USD Returns	0.0026	-0.0270	-0.0554	1.0000

Since the applications fit the data using mostly normal distributions, we chose to use the Markowitz mean variance optimization method as a benchmark for the results at various numbers of trials (10k, 50k, and 100k).

Simulation testing parameters used for each package

3.2M	**20**	**10K, 50K &100K**
TRIALS TOTAL	TESTS PER TRIAL SET	TRIALS

Benchmark Results

The first step in our analysis is to setup a theoretical result using the classic matrix math proposed by Markowitz (1952). The results are listed in the table below and will be used to calculate accuracy of the various contenders. We ran the theoretical model in Excel, R and Julia. Excel and related add-ins use a slightly different implementation of the inverse normal distribution but the error is at worst, in the highest percentile, 0.06%, thus insignificant. Nevertheless we averaged out both methods to get a fair comparison when assessing Error (Simulated Result – Theoretical Result).

When deciding on the tests, we considered the various ways to go about it and what we felt was important in this iteration of the NFS study. One method was to simulate returns and fit them while another would have been to fit the actual data while yet another would have been to use an empirical CDF (cumulative distribution function) also known as an Ogive. Of course, no matter how you cut it, if you are fitting data then you will introduce fitting error because of the different methods each packages employs. In our tests, when we fitted the data we noticed that some packages ranked different distributions as being the best fit than others. As for the Ogive, that would have been a valid approach but in the end it became logical to apply normal distributions to all the asset classes just as we would when using the calculated Markowitz method. Essentially, we calculated both the mean and the standard deviation for each data set and used those parameters to configure their [asset class] respective normal distribution. By doing this all the packages are now on equal footing.

MEAN RETURNS	STANDARD DEV	VARIANCE
0.42%	**4.02%**	**0.16%**

An important dimension to the problem is correlation. The Markowitz model uses covariance and matrix math to calculate the correlated mean and standard deviation of the portfolio. To illustrate the impact of correlation, we have used Crystal Ball to simulate 100K trials both correlated and uncorrelated. It is easy to note that the calculated results and the correlated simulation results are close enough but not so for the uncorrelated results. Clearly, the difference between the correlated and uncorrelated percentile values increases as we move away from the 50th percentile or Median. If we run this model without taking into account the correlation, we would be underestimating both the lower bound (loss) by almost 5 000$ and the upper bound by almost 10 000$. The consequence is that we are making decisions on the wrong range and risk profile – which could potentially be very good but mostly very bad because we are underestimating the downside risk.

Percentile	Returns	Calculated Value	Sim 100k w/correl	Sim 100k w/o correl
1.00%	-8.9%	$227,682	$228,153	$233,496
5.00%	-6.2%	$234,530	$234,789	$238,689
10.00%	-4.7%	$238,181	$238,424	$241,444
50.00%	0.4%	$251,060	$251,059	$251,020
90.00%	5.6%	$263,932	$263,575	$260,689
95.00%	7.0%	$267,589	$267,156	$263,443
99.00%	9.8%	$274,438	$273,969	$268,506
99.90%	12.8%	$282,114	$282,392	$274,504
99.99%	15.4%	$288,433	$289,230	$279,835

> *"Ignoring correlation means you are making decisions on the wrong range of information!"*

List of Software Tested

Solution Family	Package Name	Version Tested
Excel Add-In	Oracle Crystal Ball (Normal Speed)	11.1.2.4.6
Excel Add-In	Oracle Crystal Ball (Extreme Speed)	11.1.2.4.6
Excel Add-In	Palisade @RISK	7.6
Programming Language	Julia Language	1.0.1
Programming Language	Open R [CRAN]	3.5.1
Programming Language	Microsoft R [MRAN]	3.4.3

Test Environment

Our test machine consisted of a Dell Precision T7600 system with Dual Xeon Processors with a total of 12 cores, 16 GB of RAM, Windows 10 and Microsoft Office 365. A few notes on the test environment and simulation in general

- Most applications are single threaded. That means that one worker process runs on one processor core. For example, on a machine with 12 cores, you will get a max of 1/12 of your processor capacity. In general fewer but bigger cores make a big difference. The I series by Intel makes good use of this.
- All the applications were tested in their 64bit versions.
- Though Extreme Speed was deprecated in Crystal Ball 11.1.2.4.8, we have included the results for indicative reasons.
- We tested both CRAN and MRAN versions of R.

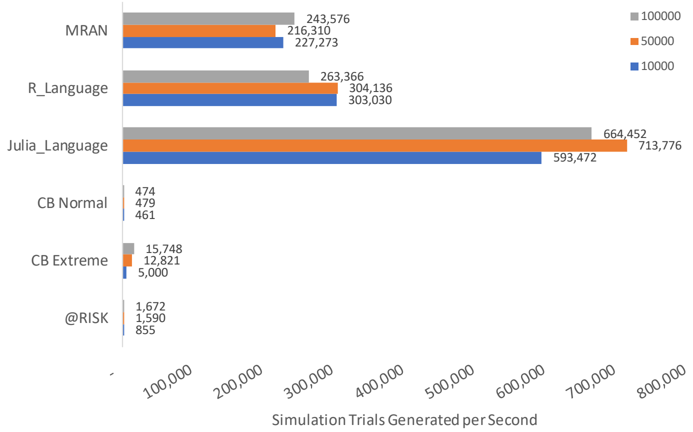
BOUT SPEE

ANALYTICS IS ALL ABOUT SPEED

HOT
NASTY
BADASS
SPEED

SPEED RESULTS

Trials per Second

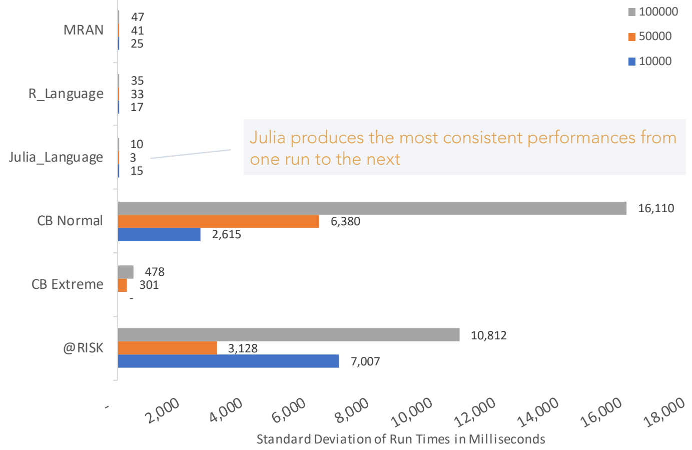


Julia blew the doors off the competition for calculation speed

Julia is 2x faster than R and 750x-1500 times faster than Excel Simulation. It is worth noting that Excel's approach to recalculation is to recurse through the entire spreadsheet whereas R and Julia focus processing on the number generation and running the calculation only on what is needed thus slashing the simulation's overhead and enhancing speed exponentially.

Interestingly, though MRAN is considered a high performance version of R, it produces 20-25% less trials or output than its Open counter part. On the other hand, MRAN can be configured for multi-threading while Open R cannot.

Consistency of run times across simulations



Julia produces the most consistent performances from one run to the next

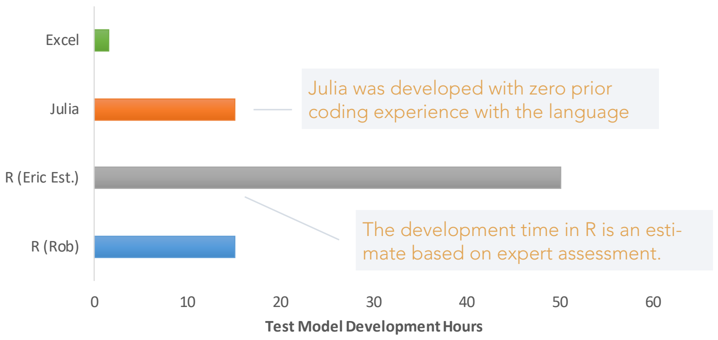
Julia is more consistent in its run-times

Julia and R both have reasonable standard deviations in milliseconds but R is still 2x to 5x more, depending on the implemetation you use. The Excel times are more erratic because Excel competes for resources with the other applications which can slow down looped processes if a task is triggered in the background, this can make a big difference in run times.

Development Time

Speed is as much about ***time-to-solution*** as it is about ***run-times***

The R Test was developed by co-author Rob Brown who published a book on the topic. It took Rob 15 hours as an expert to develop his code versus my 15 hours to develop in Julia as a total beginner! It is well known that R has a steep front end learning curve, therefore we cobbled together an estimate of what it may have taken me to develop the R code on my own had Rob not been there to do it. As for Excel, the upfront time investment is insignificant but updating is a considerable chore compared to just updating the input data and running a script as you would with R or Julia. *We will cover the economics of code versus Excel later on in the white paper.*



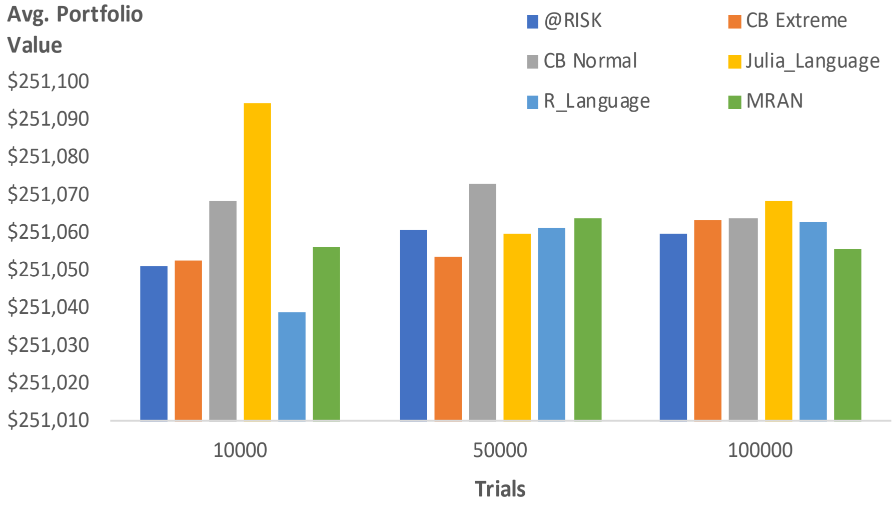
Julia was developed with zero prior coding experience with the language

The development time in R is an estimate based on expert assessment.

ACCURACY RESULTS

The global results are consistent with what was expected from an accuracy perspective. Granted there are some noise/differences that could be due to floating point error, how an algorithm was implemented, sampling error, etc. We have included graphs and a table to assess how these tools performed amongst themselves.

Mean portfolio results across all 20 simulations



Results converge as you increase trials

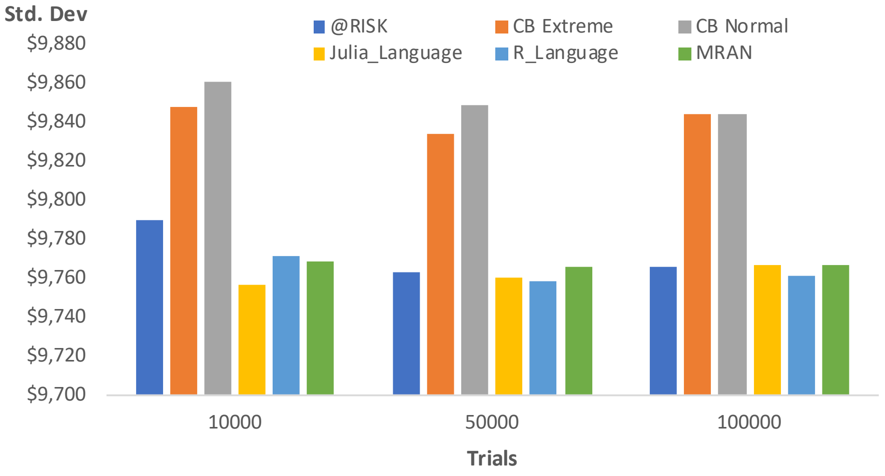
When analyzing the mean results of the simulations, it's a fairly obvious conclusion that the results of all the packages converge on the theoretical mean. Its more of a question of who gets there sooner.

The Excel Add-Ins seem to perform well at 10K trials and are generally closer to the theoretical which is important if performance to run trials is a constraint.

Technically speaking the difference among packages is close enough that it would not influence the outcome of a decision.

Standard deviation across all 20 simulations

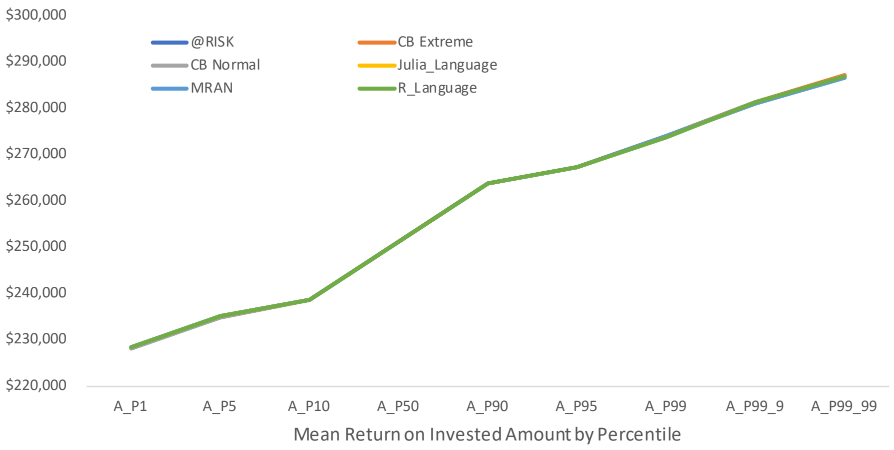
Crystal Ball has the highest standard deviation in the group



It's clear from our analysis that Crystal Ball has about 1% more standard deviation than the groups average. On the surface it appears that CB is injecting a little more risk in the sauce than its peers resulting into more trials further out in the tails. This could be accounted for because the sampling methodologies being different. @RISK, Julia and R use *Mersenne Twister* whereas Crystal Ball uses a *Multiplicative Congruential Generator.* The fact that the samples are more spread out may actually be a good thing when running less trials. Something to consider in your next model.

Mean portfolio value across percentiles

To assess a tools accuracy, we looked at the average and standard deviation at specific percentiles. Had we included 0.1% and 0.01% in our study, we would have had a symmetrical parabolic looking curve. For each trial group, we ran 20 simulations per package. This is not a tremendous amount but enough to get a sense of how accurate a package is given that we are running unseeded tests. In our opinion, you can interpret accuracy in 2 ways. Firstly, you can average out the percentiles and how consistent the results are against the peer group and its average. The second measure is to look at the standard deviation of the results to see which are more erratic across simulations at different trials (more in the next section).

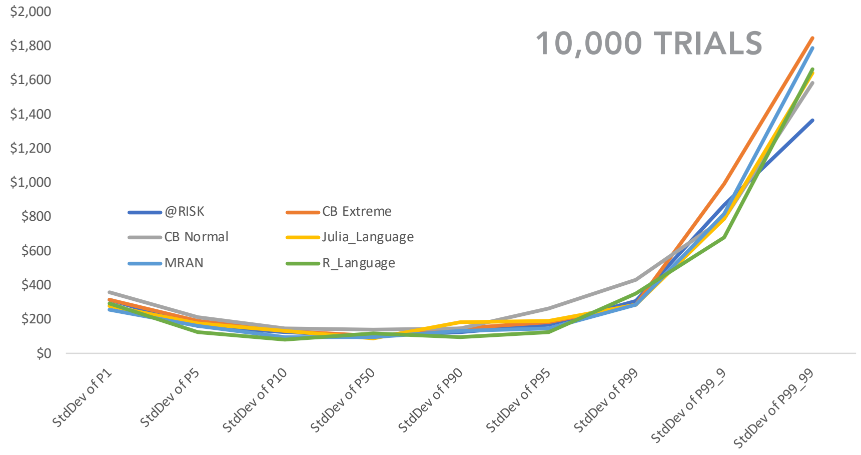


***Being that the results are so close to each other, the lines for each packages' mean line up almost perfectly.** The mean differences between packages at the highest percentile is 0.045%. So from an accuracy perspective, if this level of error is acceptable, all the tools are fairly accurate from a mean perspective, which is normal (because this is how we fall pray to the flaw of averages), thus the need to also look at the standard deviation to get a full perspective.*

Mean results by percentile for 20 simulations (by package) using Normal Distributions [Gray cells are group averages]

TEST	P1	P5	P10	P50	P90	P95	P99	P99.9	P99.99
10,000 Trials	**$228,330**	**$234,969**	**$238,495**	**$251,058**	**$263,622**	**$267,181**	**$273,889**	**$280,998**	**$286,181**
@RISK	$228,344	$234,946	$238,457	$251,046	$263,595	$267,117	$273,841	$280,990	$285,964
CB Extreme	$228,318	$234,920	$238,442	$251,051	$263,744	$267,285	$273,975	$280,970	$286,723
CB Normal	$228,121	$234,822	$238,417	$251,067	$263,660	$267,260	$273,986	$281,233	$286,036
Julia_Language	$228,498	$235,077	$238,579	$251,109	$263,597	$267,167	$273,840	$281,030	$286,513
R_Language	$228,295	$235,010	$238,532	$251,032	$263,562	$267,102	$273,887	$281,062	$286,233
MRAN	$228,404	$235,042	$238,542	$251,040	$263,575	$267,155	$273,808	$280,703	$285,616
50,000 Trials	**$228,294**	**$234,974**	**$238,514**	**$251,059**	**$263,605**	**$267,168**	**$273,818**	**$281,177**	**$287,031**
@RISK	$228,326	$235,024	$238,555	$251,051	$263,581	$267,110	$273,762	$281,109	$286,855
CB Extreme	$228,174	$234,876	$238,436	$251,054	$263,648	$267,220	$273,942	$281,316	$287,184
CB Normal	$228,150	$234,865	$238,430	$251,073	$263,689	$267,262	$273,947	$281,281	$287,030
Julia_Language	$228,317	$235,033	$238,564	$251,057	$263,565	$267,153	$273,768	$281,076	$287,273
R_Language	$228,396	$235,026	$238,549	$251,055	$263,582	$267,125	$273,689	$281,147	$287,079
MRAN	$228,401	$235,017	$238,552	$251,062	$263,566	$267,140	$273,801	$281,135	$286,765
100,000 Trials	**$228,303**	**$234,951**	**$238,507**	**$251,062**	**$263,610**	**$267,170**	**$273,828**	**$281,292**	**$287,308**
@RISK	$228,349	$234,997	$238,546	$251,052	$263,576	$267,141	$273,767	$281,222	$287,383
CB Extreme	$228,210	$234,878	$238,439	$251,063	$263,678	$267,256	$273,962	$281,588	$287,592
CB Normal	$228,173	$234,847	$238,440	$251,064	$263,671	$267,254	$273,953	$281,441	$287,299
Julia_Language	$228,349	$234,988	$238,546	$251,072	$263,580	$267,118	$273,785	$281,083	$286,973
R_Language	$228,377	$235,003	$238,546	$251,062	$263,581	$267,119	$273,739	$281,250	$287,393
MRAN	$228,364	$234,993	$238,523	$251,057	$263,576	$267,130	$273,761	$281,170	$287,208

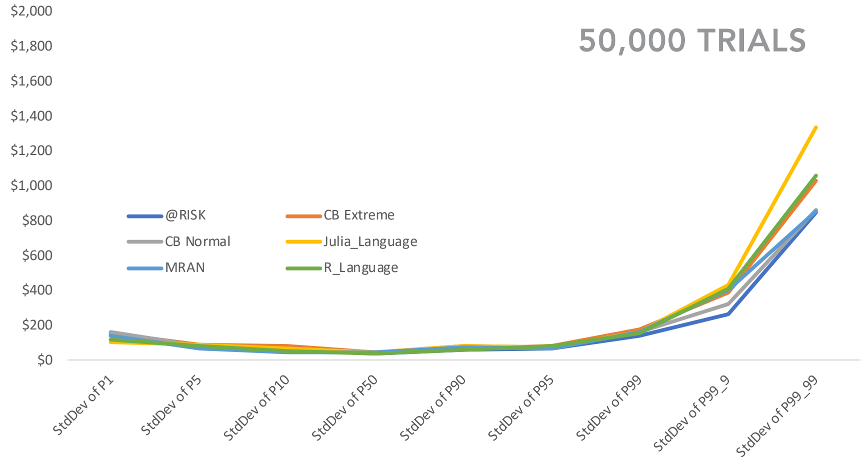
Standard deviation across percentiles

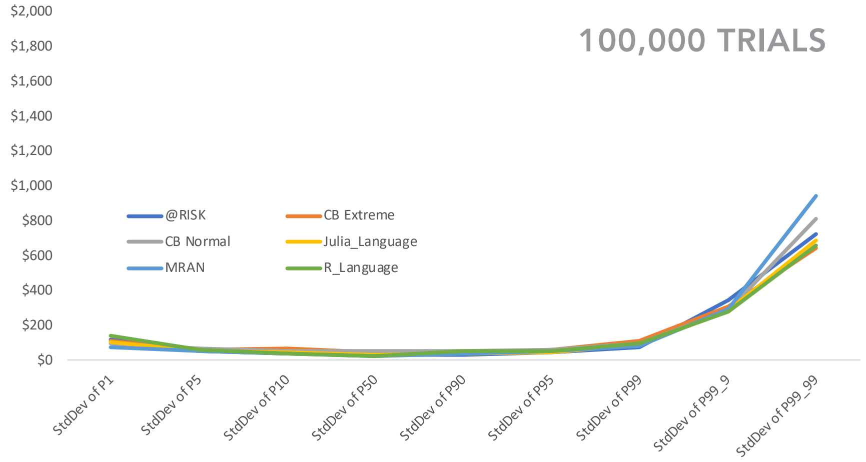


How to understand the results

In order to make sense of the results below, you need to keep in mind that the less trials you have, the more sparse the results in the tails will be. Palisade provides a good explanation in their documentation.

The Central Limit Theorem tells us that the expected variation of the mean is about ±2 times standard deviation divided by square root of number of iterations. But the relationships are not so simple for percentiles. And the further out you get, toward the zeroth or 100th percentile, the more variation you expect to see because most of the data are close to the center of the distribution, which means that out in the tails it is relatively sparse. When you get out to the min or max, you are looking at one iteration. That might be nominally a statistic of the simulated distribution, but really it's just whichever iteration happened to be lowest or highest in a particular run. In a normal distribution, or any other distribution with a tail, these are essentially meaningless numbers because the true min and max are ±infinity.



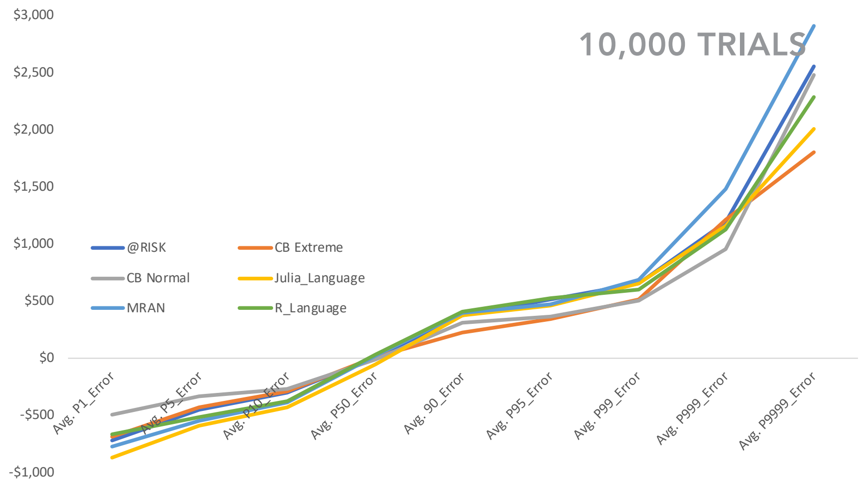


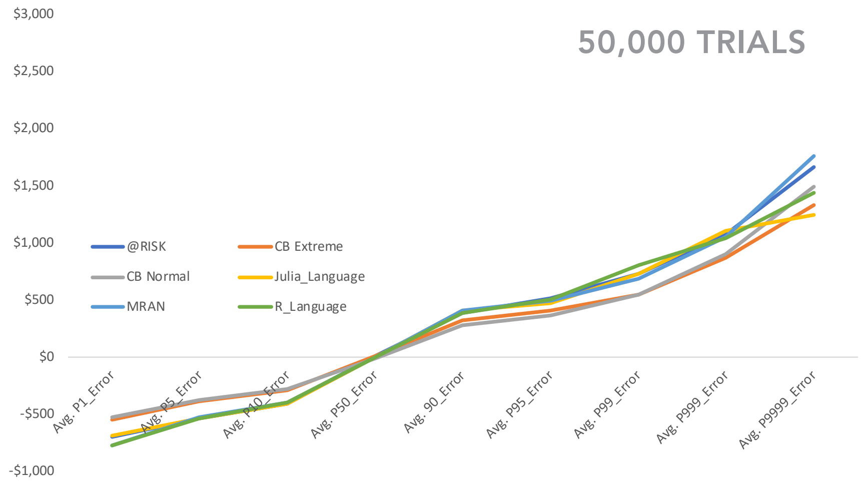
Standard Deviation decrease with number of trials

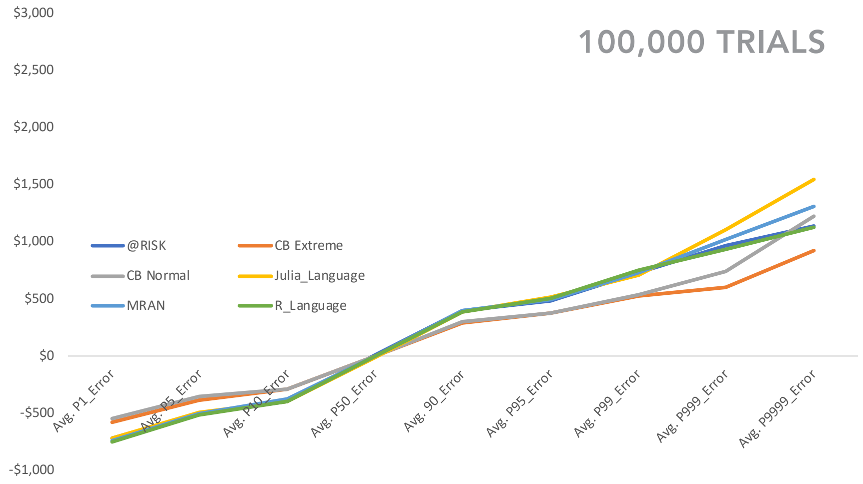
Depending on the number trials and the low number of tests (because of the Excel recalculation performance, we were limited to 20 tests per trial set), the package with the more erratic results changes as you increase the trials. It is worth further testing to see why MRAN had such different results.

At 100,000 trials the packages are producing very comparable numbers.

PRECISION RESULTS







How to understand the results

To assess the precision of the packages at different levels of trials, you only need to take the simulated percentile and subtract its theoretical equivalent. The theoretical baseline was derived by averaging the calculated results from all the packages using an inverse normal distribution with the mean and standard deviation calculated using Markowitz (1952). It was necessary to average these results because R and Julia use one algorithm for the inverse normal, while Excel and its add-ins use another. Though the differences are minor, we still needed a baseline that was common to run the test fairly. *Therefore our approach to compare precision is to see how much the simulated results differ from run to run against what is calculated and allows us to identify which packages will be closest to their theoretical baselines at lower trial counts .*

Analysis of results

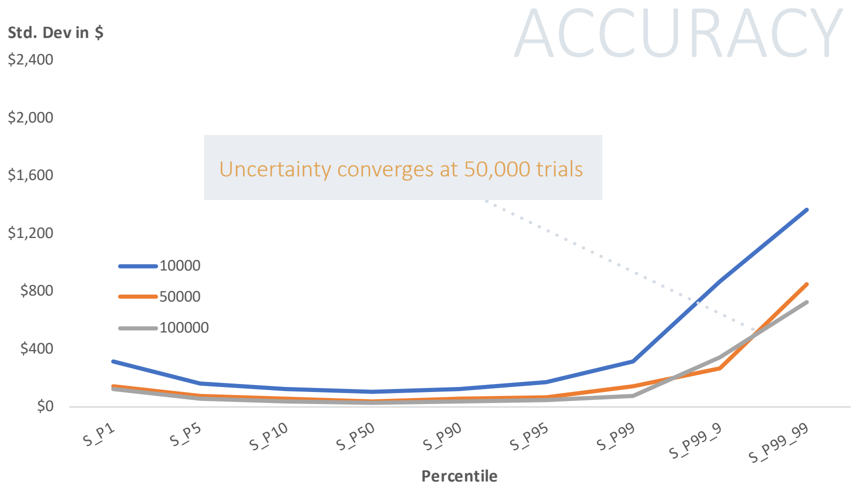
- *At 10,000 trials, what one can consider a minimum when using pure Monte-Carlo sampling yields an error at the P9999 (across 20 simulations) of about 1800 to $2950 or about 1%. MRAN had the highest error with an average (at 10k) of almost $3000 while CB Extreme speed and Julia are around $2000*
- When you bump the trials to 50,000, this almost cuts the error by up to 50% across all packages and percentiles. Though the error is not major in this context, it is worth noting that Julia once again is the top performer while MRAN though far closer to its peers than in the previous test, is still the one with the highest error in the group.
- As expected, by increasing the trials again to 100,000, we see a further reduction in error. Worth noting that Julia went in the other direction and that would be worth further investigation given we are only sampling 20 tests and coded solutions can easily produce far more. So stay tuned.

RESULTS BY PACKAGE

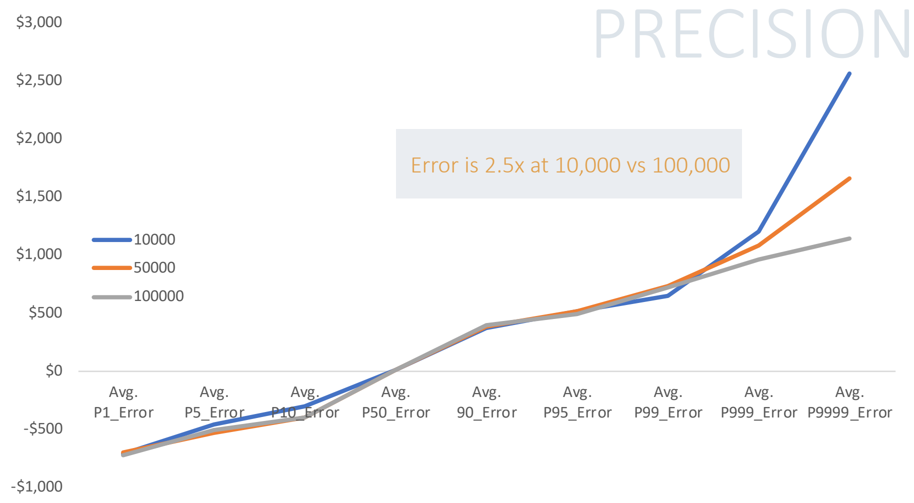
PALISADE @RISK 7.6 [EXCEL ADD-IN]

1,554	34m 20s	1h30	$1,870 US
AVG. TRIALS / SEC	TOTAL TEST TIME	MODEL DEV. TIME	COST

Palisade @RISK is considered to be the first spreadsheet add-in to add simulation capability. Originally released for Lotus 123, it has been available on Excel since the early 1990's thus making it a very longstanding product. The tool is very complete and offers a wide variety of analytics tools that allow you to solve most one off problems.



Uncertainty converges at 50,000 trials



Error is 2.5x at 10,000 vs 100,000

What we like

@RISK has a very complete set of tools that covers pretty much anything you may want to tackle in excel, including time series modeling, decent optimization tools which include OptQuest and importantly for MS project users, the ability to simulate schedules.

What gave us heartburn

One of @RISK's greatest assets is also one of it's greatest difficulties, its code base. As Palisade @RISK has evolved in sophistication and speed, so have the stability issues increased in prevalence. When scripting the tests, we had several issues where the automatic multithreading would kick in and most threads would crash. The setup time of a multi-core simulation far outweighed the time benefit, notwithstanding that most threads crash anyways and that most attempts to kill and restart the simulation on a single core (which is pretty fast by Excel standards) end up producing corrupt results because it stops applying correlation.

ORACLE CRYSTAL BALL 11.1.2.4.6 [EXCEL ADD-IN]

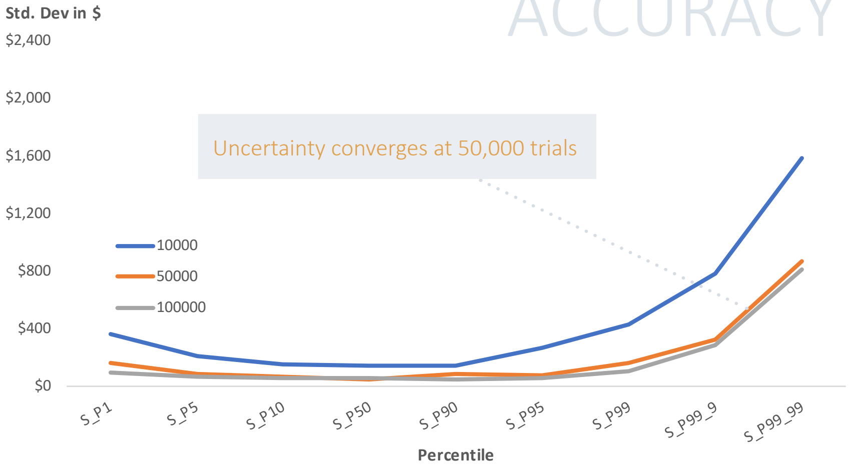
475	1h53m 20s	1h30	$995 US
AVG. TRIALS / SEC	TOTAL TEST TIME	MODEL DEV. TIME	COST

Only the Normal Speed Results are presented here as Extreme Speed is now deprecated and is here for illustrative purposes only.

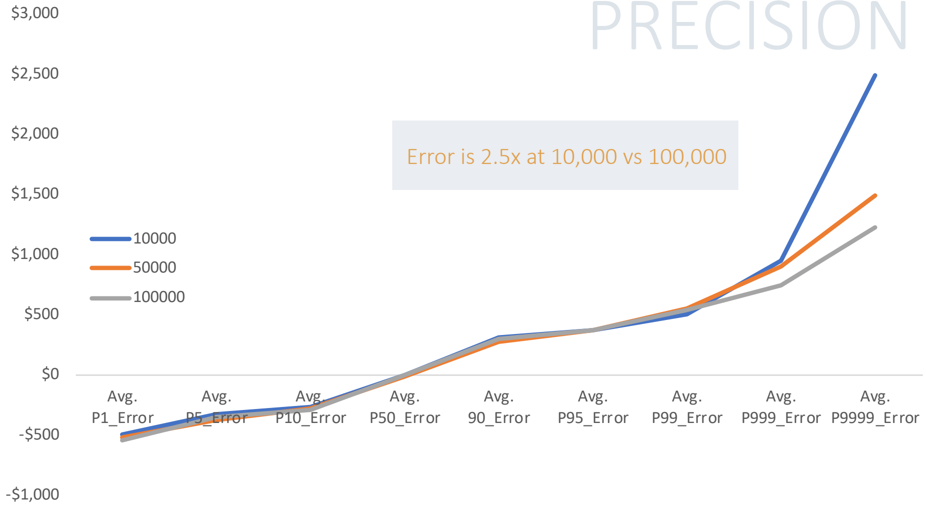
ORACLE® CRYSTAL BALL

Oracle Crystal Ball is also a venerable player Monte-Carlo simulation tool for Excel that has been around since the early 1990's. They have innovated on many levels, including introducing the first non-linear simulation/optimization (OptQuest) for Excel, now considered standard in any respectable simulation package.

ACCURACY



PRECISION



What we like

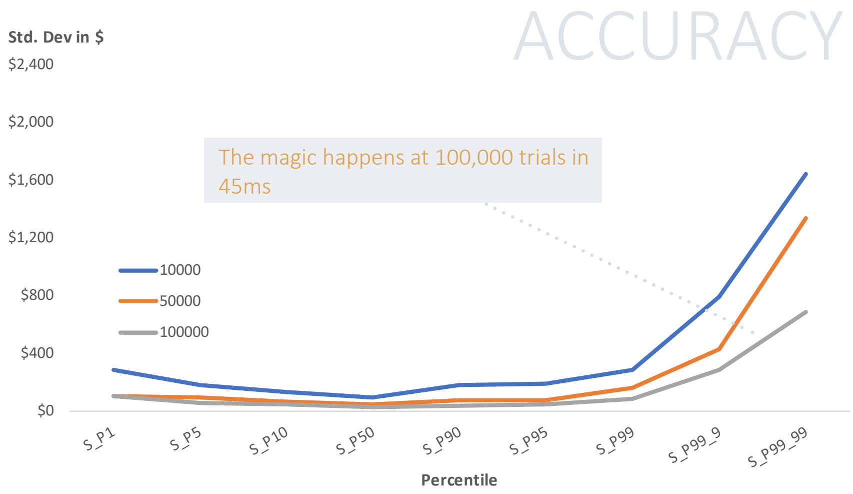
Crystal Ball was designed with usability for 80% of the users in mind. That means some of the power-user stuff is hidden a layer or 2 but that building a simulation model is easy and efficient. Crystal Ball also natively hides the simulation formulas, thus making the spreadsheets easy to share and protects the underlying logic. There are also solid time-series tools and, like @RISK, a collection of equally solid analytics tools for modeling and simulation.

What gives us heartburn

Unfortunately, Crystal Ball has been part of Oracle for over 10 years and this has had consequences on the developers' ability to deliver new features, so one can consider this a stable product going forward with minor improvements along the way. Another consequence of such a niche product becoming part of Oracle is the deprecation or removal of PSI (Polymorphic Spreadsheet Interpreter) that gave anywhere from a 10x-100X reduction in run-time. This feature allowed to run optimizations at light speed but was removed in release 11.1.2.4.8

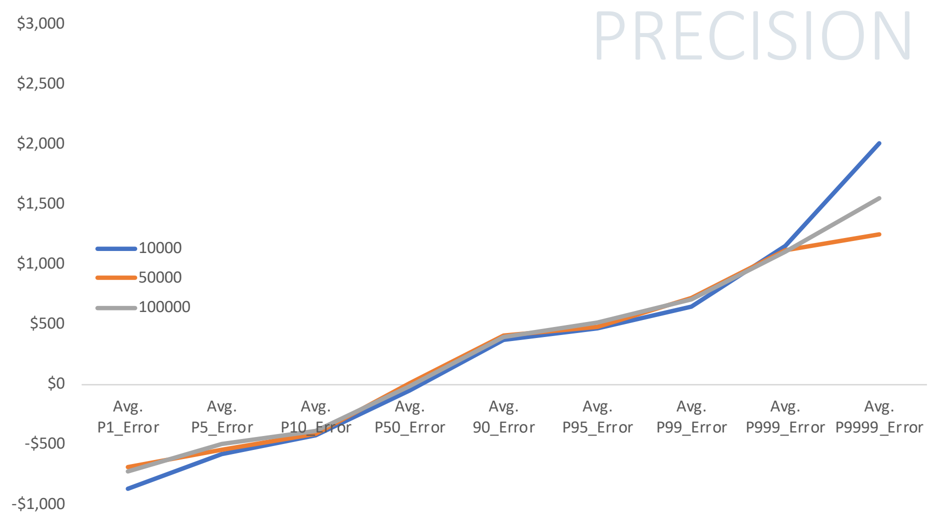
JULIA LANGUAGE 1.0.1 [PROGRAMMING ENVIRONMENT]

674,000	4.75s	15h00	FREE
AVG. TRIALS / SEC	TOTAL TEST TIME	MODEL DEV. TIME	COST



The magic happens at 100,000 trials in 45ms

The Julia Language is a new entrant that was launched officially in 2012. Its primary goal was a language that could give you the best of R and Python and be incredibly fast. More specifically, the authors wanted a language that could handle vector and matrix math in near natural language. Though version 1 was only released in September, there is a growing community of users focused on different problem areas such as AI, Machine Learning, Deep Learning, Simulation, Forecasting, etc. Ladies and gentleman, this is one to watch.



What we like about Julia

Firstly, if you have some basic skills in programming, picking up Julia is no biggie. I would add that it's DataFrames Module (Tabular Format, like Pandas in Python) and its functional syntax (the ability to code functions) makes Julia the easiest and most logical language to learn after Excel + VBA. A common situation for many analysts thinking about what comes after. It's like Excel without the sheets. Notwithstanding the speed, you can call packages from R, Python, Fortran and C quite seamlessly. This means as Julia grows, you will never be out of a package. Lastly, contrary to some other math based languages, Julia handles loops and iterations with incredible speed and ease. It avoids making a choice based on vectors versus loops – you can use both. The language is be increasingly being adopted and new packages are forthcoming everyday.

What we don't like about Julia

The language has been stabilizing for almost 4 years... that means there are more than a few syntaxes or functions that have changed names over time. When troubleshooting, an older explanation was not guaranteed to help an unexperienced user. Another beef with Julia is the lack of native packages. Though you can easily call packages in other languages, you still need to know their syntax and you will probably sacrifice speed in the process.

R LANGUAGE [PROGRAMMING ENVIRONMENT]

R is a statistical language that was first introduced in 1995. It was intended as an open source version of the S language that was developed at Bell Labs in 1976 by John Chambers. R has slowly gained traction due to its open source framework, 1000’s of academic contributors and a long list of packages. R is a very capable and complete language for statistical computing. *Currently 2 flavors of R exist: CRAN and MRAN.*

What we like about R

R is a very well documented language with a multitude of open courses, YouTube videos, books, university classes and so fourth. R has been hot in the age of analytics. There is a considerable number of packages that do almost anything that can be installed via a few lines of script. Also, most vendors have incorporated some form of integration that allows you to inject your R code into production. Lastly there are some good IDEs, including Visual Studio and RStudio.

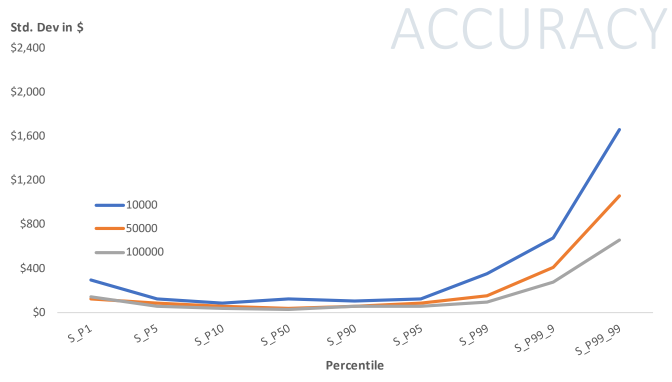
What gives us heartburn

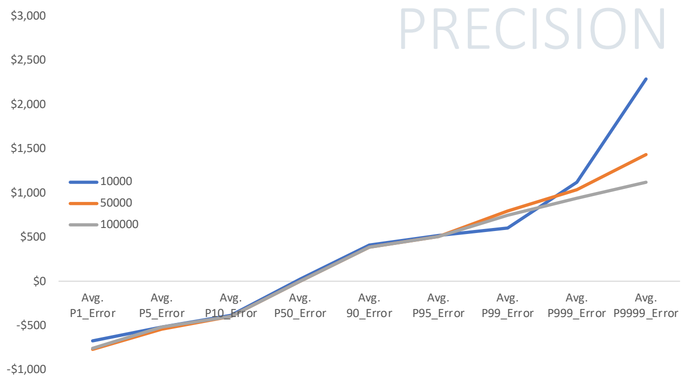
Though my technical co-author is the real expert on R, I need to point out that he does not agree with me that the syntax can be burdensome at times with directional operands, inconsistent indexing structures, and being forced to use apply/sapply/lapply, all of which can introduce lots of confusion to the uninitiated. Scaling the R language is not an easy task and requires lots of outside help. Dynamic Typing and the ability to handle looping *efficiently* are lacking.   Modules are buyer beware in regards to algorithmic or processing errors. In fact its almost a good idea to make to test the output before adopting a new module. This is the benefit of having a community curated package versus one from an individual.

Open R (CRAN) v3.5.1

This is the open source version. Its very well suited for single machine analysis but makes up for this by being easy [the code] to inject into production systems. Oracle has a database server called Enterprise R which enables to process R directly on the databases.

277,250	11.55s	15h00	FREE
AVG. TRIALS / SEC	TOTAL TEST TIME	MODEL DEV. TIME	COST

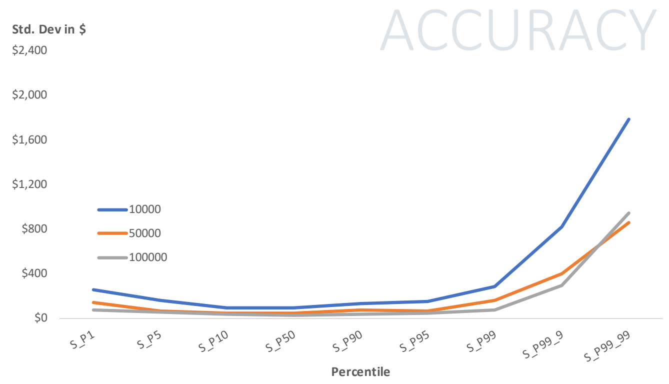


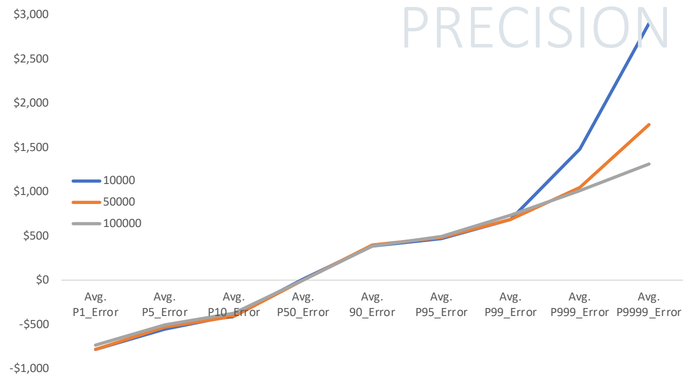


Microsoft R (MRAN) v3.4.3

Microsoft R Open a re-branded version of Revolution R Open. This is the Microsoft Version of R. It has built-in integration with Microsoft R Server and allows for running on multiple core. For this reason, we wanted to see if this implementation had any differences performance or accuracy differences with it’s open cousin.

233,340	**13.72s**	**15h00**	**FREE**
AVG. TRIALS / SEC	TOTAL TEST TIME	MODEL DEV. TIME	COST





CONCLUSION

So who won?

And the winner is…. Julia! Julia is the fastest on all dimensions (including development time for coded solutions), period! As stated over the course of this whitepaper, speed has 3 main dimensions: *processing power, learning curve* (how long it takes to pick up the language) and *time – to – solution* (development Time). We are going to look at each of these dimensions why we made the pick that we did. Afterwards, a simple framework to decide which tool or approach makes more business and economic sense.

Processing Power

Julia is more than 2x faster than R and between 750 to 1500x faster than Excel simulation. Both R and Julia are faster because the have considerably less overhead than Excel and do not need to compete as aggressively against other programs for resources. We are not the first study to benchmark Julia and our *goal was to contrast Julia to existing simulation tools* rather than calculate each solutions' raw power. If you want more technical benchmark data, we recommend visiting the NASA website (https://modelingguru.nasa.gov/docs/DOC-2676)

Learning Curve

Notwithstanding how much faster it may be to derive a one-off answer in Excel, if you goal is to develop a fast, reusable model in a clear and easy to use programming language, than the outcome is pretty much set in Julia's favor. I [Eric] have been looking at R and Julia since 2014. I always wondered why I would use R when I had Excel and a ton of add-ins. The answer is that business case modeling in R is not as intuitive or practical as in Excel and otherwise did not provide a compelling offer for simulation and risk modeling. In fact Rob wrote a book on this very topic because nothing existed on the topic. On the other hand, Julia handles both vector math and looped structures with ease, making it a more forgiving and intuitive to learn.

Having programmed in VBA, some Pascal, BASIC, SQL and Python over the years, I am fairly familiar with programming and its constructs. Yet with all that, I had a hard time wrapping my mind around R. It always seems a little idiosyncratic and suffers from some consistency issues.

As for Julia, even though I started picking it up for this study, I have been looking at code snippets for years and each time it left me with the feeling "Hey. I can do this also!". That feeling was well founded because within a few weeks in my off-time I was completely productive in Julia. So we consider Julia faster to learn than R but more laborious than Excel.

Time-To-Solution

Julia and R had similar time to solutions if you were to compare 2 experienced users while Crystal Ball and @RISK also were quite similar. The issue is when should you code and when should you just build a quick Excel model? This is a simple business problem focused what is best for executing analysis whose answer is always "It depends". We recommend a simple framework and break-even calculation to decide what makes more sense in your day-to-day reality.

What about @RISK and Crystal Ball?

As we mentioned, these are very good tools with lots of history for solving problems. ***Speed for these tools is time to the first answer.*** The issue is that these tools produce insight quickly but cant really be put into production as-is without porting the algorithm to Python, C# or Julia. ***The time savings are truly in situations when the problem is rare and once solved, the manager moves on to a new different question.***

Tying software's dimensions of speed to business success

Thomas Edison said it best *"Vision without execution is hallucination"* because the link between knowing what's best to do and action is fairly clear. He was ahead of his time because business execution is often ranked as the number 1 critical success factor by business magazines such as the Sloan Management Review and the Harvard Business Review. The other critical success factor for modern business is velocity/speed. Organizations that deliver faster, have the shortest cycle time for analysis, answer concerns faster, understand market patterns quicker, capitalize on opportunities more rapidly, generally outperform their peers. By focusing on these two critical success factors, firms can derive a competitive advantage over larger firms who enjoy more resources and scale.

> *"Vision without execution is hallucination"*
>
> Thomas Edison

Competitive Advantage comes from speed and accuracy of decisions

> *"Analytics strategy can only deliver either faster decisions or better ones, including a combination thereof. "*

Analytics strategy can only deliver either faster decisions or better ones, including a combination thereof. Knowing which one of these ideals is your primary goal allows you to figure out a strategy, what kind of tools / platforms to use and how much resources to allocate. From a practical point of view, analytics provides a pathway to answering questions of all kind, e.g. How much should I make of A vs. B, what day is best to advertise on the web, where should I place my next store, should I build a new plant, etc... Each of these questions, which all have a different value and utility, require answers – **usually fast!**

Getting the problem-solving approach and development strategy right will translate to faster cycle times and more opportunities for further insight. Most business questions are based on some sort of trade off or opportunity cost, so it is important to keep in mind that a dollar spent solving one problem is not going to be spent on another. Ultimately success lies in matching the most efficient and effective strategy with action to achieve results better than the other guy/gal. In that spirit, let's take a few moments and discuss when it is appropriate to code and when its better to use other tools/environments for rapid and one off decisions.

How to pick the right solution for your needs?

What are the properties of the decisions you are making?

- **Define the Value/Benefit of the decision.** E.g. Payback, ROI, Process Capability Improvement, etc. This is based on the values and economics of your business.
- **Define the cost of the decision.** This is necessary for applying utility theory and basically making trade-offs based on bang for the buck.
- **Time Horizon for taking action.** You will either have lots of time or very little to decide. This will dictate how much effort and how granular your analysis needs to be.
- **Frequency of decisions.** If a decision happens regularly, how often do you have to make it. The more frequently a decision is made, the better the ROI can be on your analytics project.

Decision Frequency

	Once	Recurring
High Risk + Value	Bespoke Decision Model	Automate using ML+AI
Low Risk + Value	Wisdom Heuristics	Automate using rules

Decision Utility

What kind of decisions are you planning to make?

- **Wisdom Heuristics** are basically using experience to make decisions.
- **Automating rules i**s about taking formal classifications like age or number of units bought and create a business rule to be applied automatically to the data during processing. This is usually a coded solution.
- **Bespoke decision models** are built when the decision is rare or unique but has a big impact on the firm. E.g. Merger and Acquisitions, opening up a new national office, building a new plant (CAPEX). Once the decision is made the model is filed away as support material.
- **Advanced Automation of Rules** can make use of pattern detection, machine learning and artificial intelligence on vast amounts of data to make decisions and recommendations automatically. These systems when well designed will evolve their predictive ability.

Assessing the economics of code versus Excel

Building and updating a Portfolio in Julia versus Excel

Both the Julia and Excel versions of our model start at the point where the data was downloaded and the log returns pre-calculated. Though a subsequent version of our model could download and process the ticker data, we elected not to use it since the Excel model did not and we wanted to keep the playing field level.

In order to build a simulated Markowitz Model you need to:

1. Download the data for all the tickers using CSV (2min/ticker)
2. Consolidate Asset/Ticker Data into one table for processing
3. Calculate the LogReturns for each asset
4. *Calculate the correlation and covariance matrices*
5. *Update tables and matrices to reflect new number of assets*
6. *Run Model*

Metric	Julia	Excel + @RISK
Dev. Time	15hrs	1.5hrs
Update Time	1min	15min
Run Time	151ms (100K)	58,900ms (100k)
Updates/Hr	60x	4x
Max Sim/Hr*	24,000x	60x

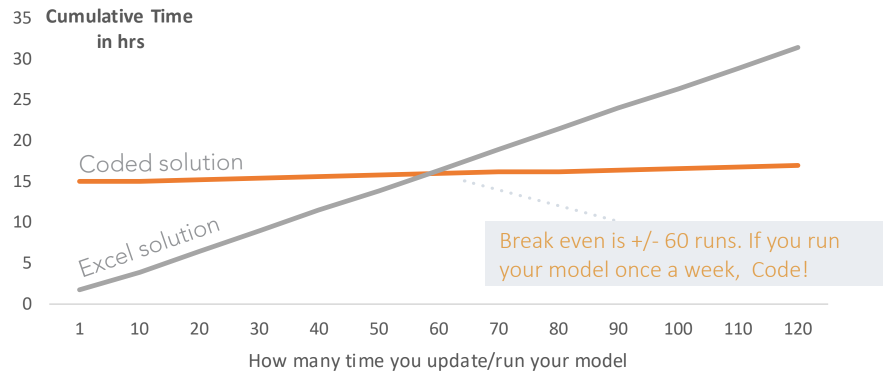
*Max number of simulations an optimization can run in one hour

Our timing test focused on adding one asset class or ticker to the model and re-running (steps 4-6 above). In the case of a portfolio model that needs to be run many times, the answer in a no brainer. But in reality it's more subtle. **Under these circumstance, the economics and ROI are clear with a simple break-even calculation that states running this model more than once a week would justify the investment in a coded solutions that is also less error prone upon updates.**

Using the numbers provided above, we see that a coded solution with 15hr of development time breaks even with Excel at 60 runs. However, to update the Excel Model, you invariably touch the model's logic and structure which in our opinion can lead to correcting old errors *while introducing new ones* because ***studies have suggested that the initial probability of a significant error in your Excel model is about 90% (Harris, 2017)***. On that basis, ***what is not calculated in the update time****, which could significantly reduce the break-even, is the additional testing you need to perform every time you update your Excel model and its data.* Whereas a coded solution that is properly tested should not require more than a data update thus mitigating the risk of introducing new errors into the code which avoids a lot of lost time diagnosing and testing results.

Granted, much can be done to automate this but we are strictly comparing our models in their current state. To make the functionality of Excel as automated as our Julia version, the Dev time would be far closer to 15hrs instead of 1.5hrs due to the need to incorporate some VBA. Say you manage to develop an equivalent Excel VBA solution, ***you will never have the same raw power, speed, reliability or scalability.*** If this is the case then Julia is the better option from day 1 for this type of model.

The Economics of Coding Analytics



Break even is +/- 60 runs. If you run your model once a week, Code!

So how to decide which tool makes sense?

Here is our take on it. If you are only making a decision once, go for what is fast and generally reliable such as an Excel model, a pen & paper, gut instinct, etc. Though you may make less optimal decisions on the spot, over time you will win out because of speed or better yet, time to decision. E.g. Saving 48% on your new equipment if you buy today instead of a month from now needs an immediate and unique answer. On the other-hand, when a decision gets made over and over again, coded solutions that are well tested are the way to go for repeatability and speed of execution (time to decision). Given that coded solutions take more time and consequently cost more money to develop than certain types of bespoke models, the time ROI accrues every time you run your coded models.

In terms of accuracy, studies have proven that computers will out perform humans on repetitive decision tasks such as loan approvals, school admissions... The main reason for this is that coded computer algorithms don’t have the inherent biases of humans and will apply rules and approvals consistently. Now I should point out that if you build in biases into your models by accident, usually a risk in machine learned models, you will have biases in your decisions. The main point to keep in mind is that a machine will apply your decision criteria consistently, well thought out or not.

Decision Frequency

	Once	Recurring
High Risk + Value	Analyst Modeling Tools (eg. MS Excel, Analytica, Project, Visio...)	Machine Learning + AI with Model Management
Low Risk + Value	Quick Excel Model or Experience	Case Based Decision Rules

Decision Utility

Lastly, though there are many ways you can segment your decision model development projects, keep in mind the utility and frequency of decisions are essential and should be a the root of any analytics budget/allocation strategy.

REFERENCES

- Eric Torkia (2019), “To code or not to code, that IS the question! [for analytics strategy planners]”, published on January 14, 2019, (https://www.linkedin.com/pulse/code-question-analytics-strategy-planners-eric-torkia-masc/)

- Robin Harris (2017) “Excel errors: How Microsoft’s spreadsheet may be hazardous to your health”, July 28, 2017, https://www.zdnet.com/article/excel-errors-microsofts-spreadsheet-may-be-hazardous-to-your-health/

- Wikipedia contributors. (2019, January 7). “Accuracy and precision”, Wikipedia, Retrieved 01:57, February 7, 2019, from https://en.wikipedia.org/w/index.php?title=Accuracy_and_precision&oldid=877182167

- Harry Markowitz (1952) “Modern portfolio theory”. Wikipedia, Retrieved 01:59, February 7, 2019, from https://en.wikipedia.org/w/index.php?title=Modern_portfolio_theory&oldid=878668017

- Ronald L. Iman & W. J. Conover (1982) A distribution-free approach to inducing rank correlation among input variables, Communications in Statistics- Simulation and Computation, 11:3, 311-334, DOI: 10.1080/03610918208812265

AUTHORS

Author
Study Designer

Eric Torkia is the executive partner and analytics practice lead at Technology Partnerz Ltd., a firm specialized in delivering analytics solutions focused on forecasting, simulation and optimization to organizations all over the world. As practice lead, he is certified on multiple tools (Oracle Crystal Ball, Oracle Primavera Risk Analysis, Vose ModelRisk, Palisade @Risk, Frontline Risk Solver) and is often consulted on analytics strategy planning and executions by senior executives.

etorkia@technologypartnerz.com

Technical Co-Author
The R Expert

Robert D. Brown III is an internationally recognized author and consultant specialized in the art of decision science using Analytica, R and Excel to name a few. Rob was instrumental in deriving the testing algorithm and deploying it in R. Many thanks go out to him.

On a normal day, Rob's main drive is to help his clients measure the value and the risk associated with the important decisions they face in order to make informed trade-offs and choices. If you don't believe it check out Rob's most recent publication is Business Case Analysis with R through Springer-Nature-Apress.

rdbrown@technologypartnerz.com

TP
TECHNOLOGY
PARTNERZ

Our Sponsor

Established in 1999, Technology Partnerz is a management consulting and technology firm focused on the rapid adoption of predictive analytics tools, practices and results within the organization by providing strategy, business analysis and organizational change management services to its clients.

info@technologypartnerz.com
+(514) 278-2221

TECHNOLOGY PARTNERZ

Questions answered,
Problems Solved!