This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2021-9726C

Sandia National Laboratories

Exceptional service in the national interest

Sandia's Uncertainty Calculator

A case study in the graded approach to software quality assurance

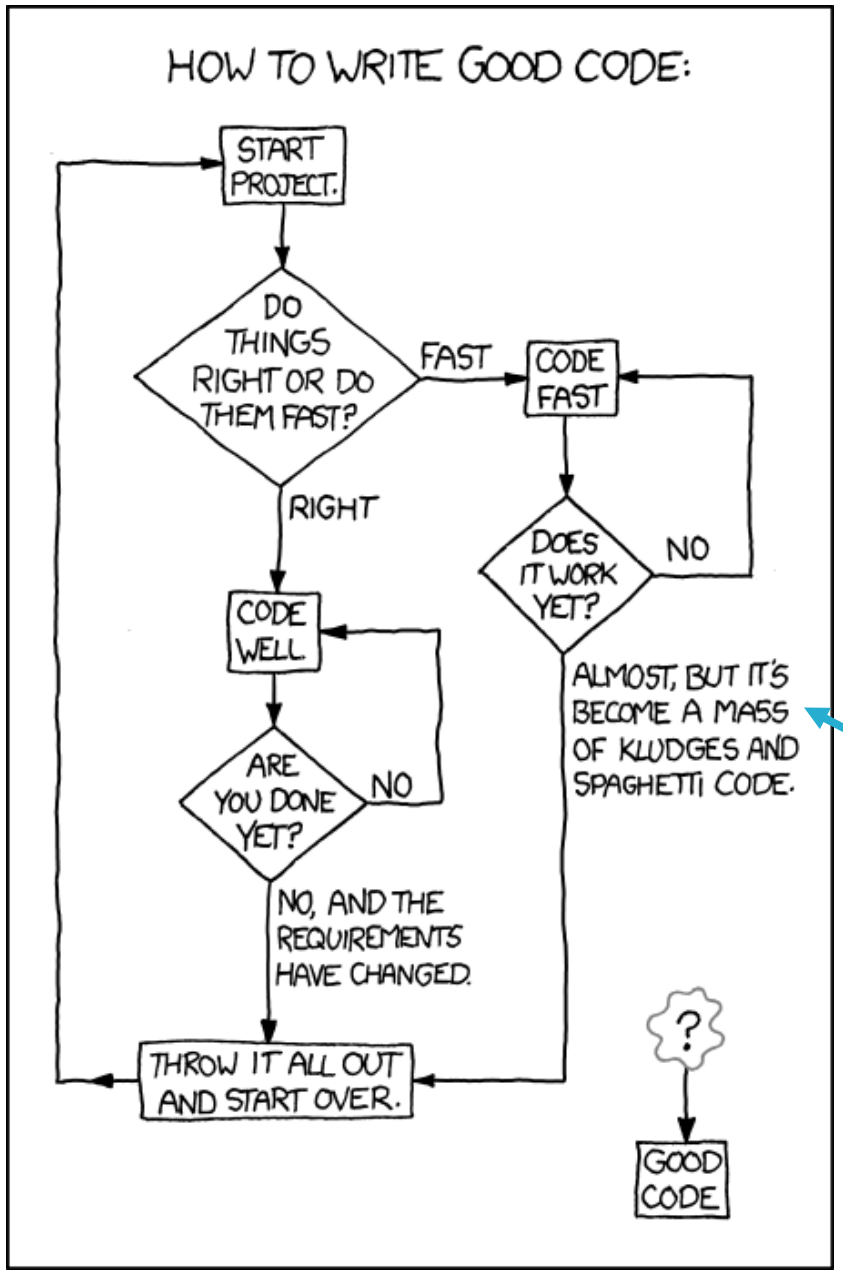
Collin J. Delker

NCSLI Workshop & Symposium

Orlando, Florida, 2021

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.



https://xkcd.com/844/

Why this talk?

- Software validation is required by ISO/IEC 17025:2017, with little guidance on how to implement
- Most metrologists are not software engineers, yet custom in-house metrology software is common

Suncal is somewhere in here

Outline

- Overview of Suncal Software
- What is software quality?
- Graded approach to SQA
- Stages of Suncal software evolution and SQA

Sandia Uncertainty Calculator (Suncal)

Calculations:

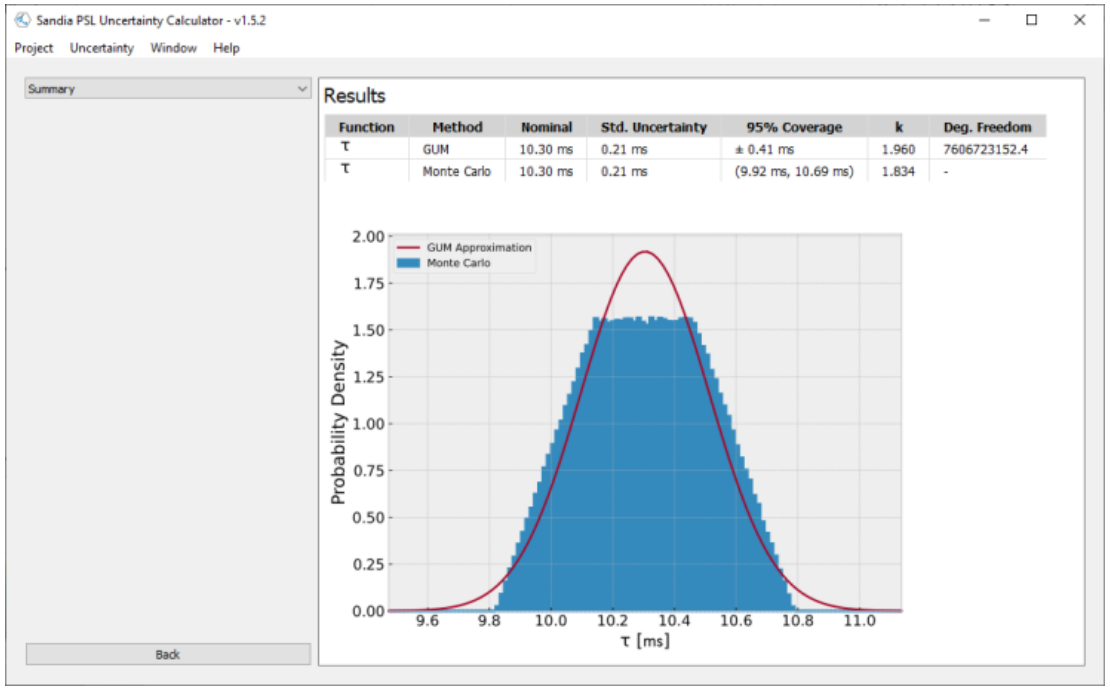
- GUM (symbolic) and Monte Carlo Uncertainty propagation
- Uncertainty in linear and non-linear curve fitting
- Measurement Decision Risk
- Calibration Interval Analysis
- Analysis of Variance, autocorrelation

Usage:

- Windows/Mac/Linux user interface
- Command-line interface
- Python library

Free, open-source, GPL License

- https://sandiapsl.github.io



Motivation for developing Suncal

But what about _____ commercial software that does uncertainty calculations?

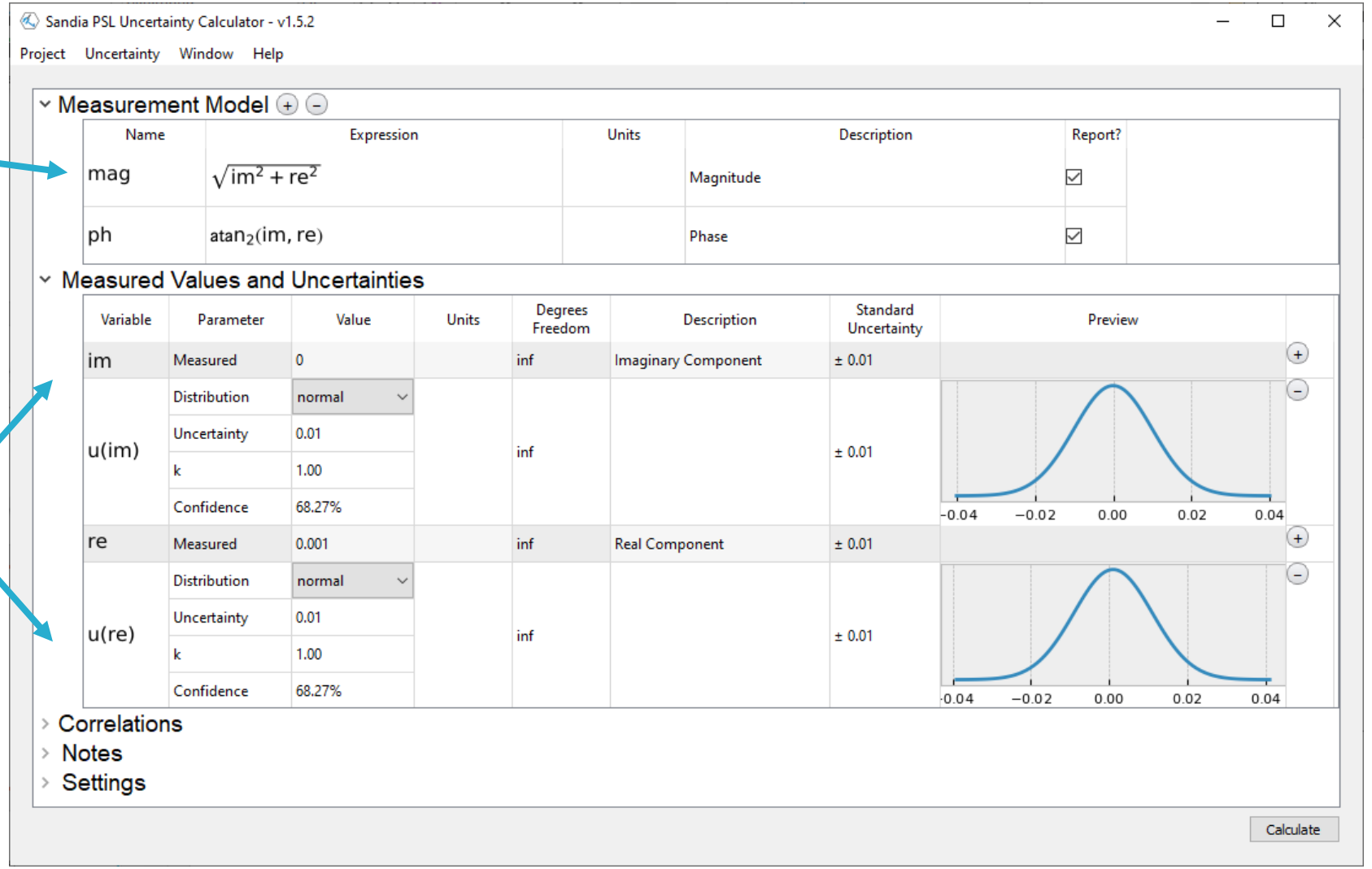
While other uncertainty software exists, we found it to be too limiting:

- No symbolic solutions to GUM equation
- Limited handling of units/dimensional analysis
- Limited complexity of model, limited number of input variables
- Too much hand-calculation still required (e.g. combining Type A and Type B components)
- Limited output reporting
- No integration with other metrology calculations (risk, ANOVA, curve-fit, etc.)
- No native handling of complex numbers
- Limited extensibility
- Security

Suncal Example – Magnitude and Phase (See GUM-Supplement 2)

Enter measurement model(s)

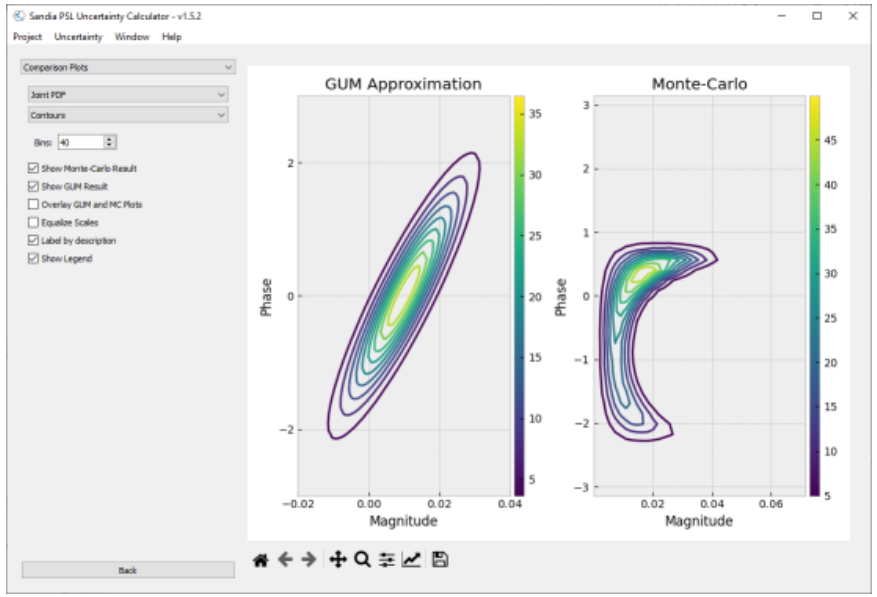
Enter values and uncertainties

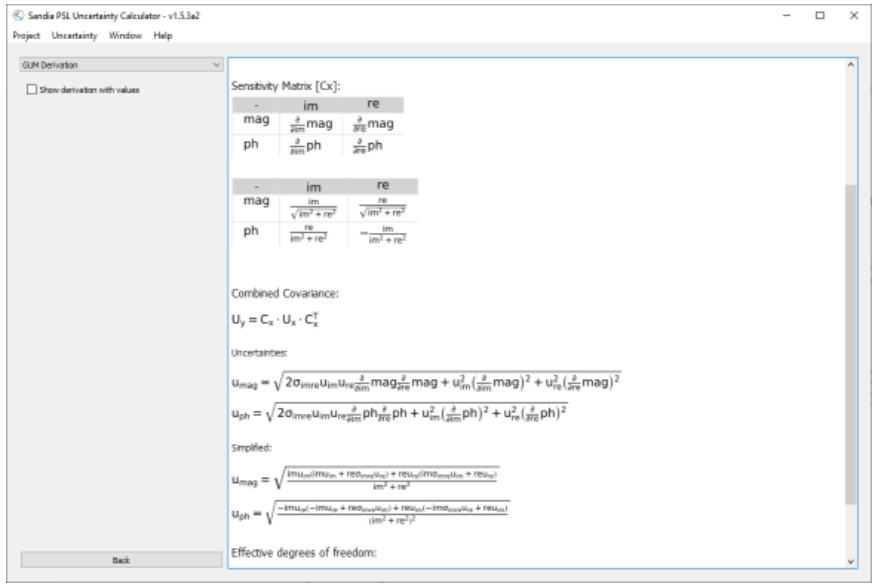


Suncal Example – Magnitude and Phase (See GUM-Supplement 2)

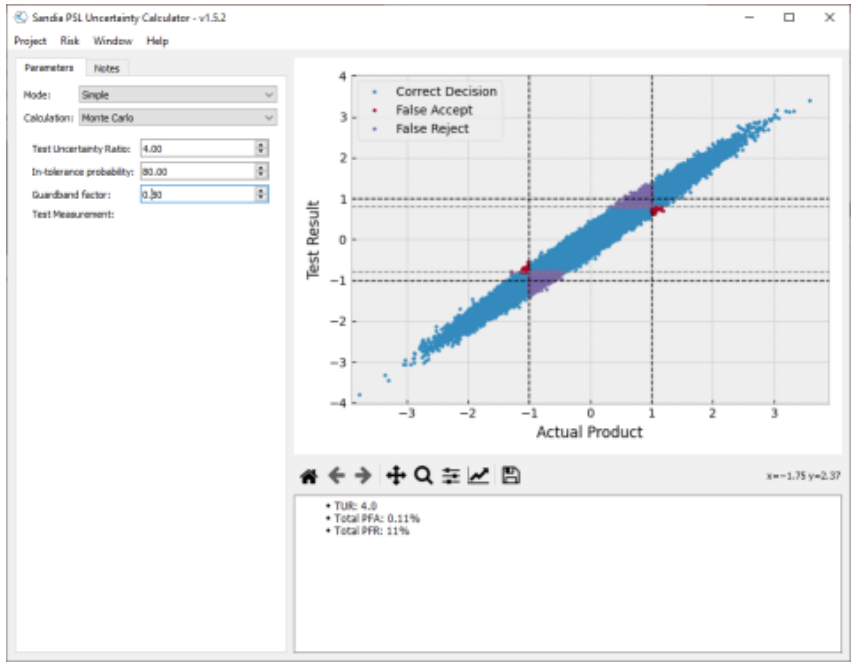
Results include:

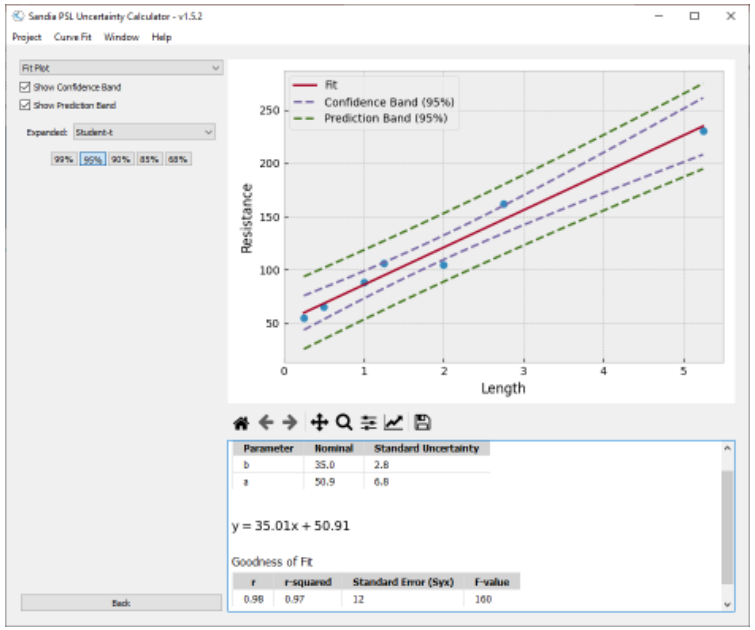
- GUM Evaluation
  - Derivation of equations
  - Sensitivity coefficients & proportions
- Monte-Carlo Evaluation
  - Convergence plot
  - Distribution fits
  - Symmetric & shortest coverage intervals
- Expanded uncertainties
- GUM and MC comparison
- Joint probability plots, correlations

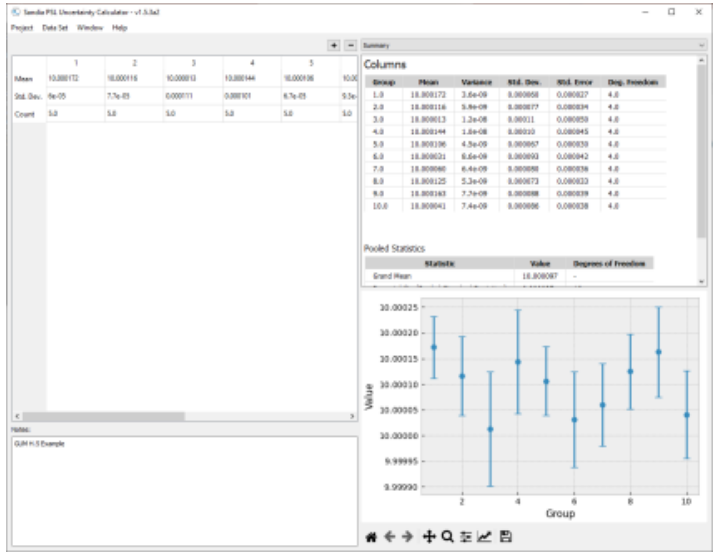




Other tools: decision risk, curve fit uncertainty, ANOVA, interval analysis…







Software Quality

Everybody can instantly recognize bad software, but how do you recognize quality software?

Some aspects of software quality assurance (SQA):

- Defined and traceable software requirements, software design document
- Version control (both code and requirements)
- Formal change request procedures
- Documented code reviews
- Verification and Validation (unit test, system test, integration test, etc.)
- Usability testing
- User documentation, training requirements
- Build and deployment procedures

Software is never finished, only abandoned.

Graded Approach

Overall risk level of software relates to level of rigor required in ensuring its quality.

- Risk level RL = consequences x likelihood
- Practice Level PL describes SQA activities required and/or suggested at each level

Policy provides details on how to determine C and L levels. Examples:

- SW failure that can result in loss of life or serious injury is C4
- Severe damage to environment contained within boundaries is C3
- Moderate damage to reputation or customer relationships is C2

Graded Risk Level (RL) Associated SSQAP Recommended Practice Level (PL)					
Likelihood Tier Undesirable event due to software failure	**Consequence Tier** Undesirable Event				
	C4 (Catastrophic)	**C3** (Severe)	**C2** (Moderate)	**C1** (Low)	**C0** (Negligible)
L4 (Very High)	RL = VH PL = P4	RL = VH PL = P4	RL = H PL = P3	RL = M PL = P2	RL = L PL = P1
L3 (High)	RL = VH PL = P4	RL = H PL = P3	RL = M PL = P2	RL = M PL = P2	RL = L PL = P1
L2 (Moderate)	RL = H PL = P3	RL = M PL = P2	RL = M PL = P2	RL = L PL = P1	RL = L PL = P1
L1 (Low)	RL = M PL = P2	RL = M PL = P2	RL = L PL = P1	RL = L PL = P1	RL = N PL = P0
L0 (Negligible)	RL = L PL = P1	RL = L PL = P1	RL = L PL = P1	RL = N PL = P0	RL = N PL = P0

Legend:
RL values: N = negligible, L = low, M = moderate, H = high, VH = very high
PL values: P0, P1, P2, P3, and P4 are defined in the Guidance to SSQAP Practice Levels. Practice activities related to these practice levels are provided in Table 3-3.

Sandia Software Quality Assurance Policy

Suncal Software Evolution

- Suncal evolved from a few lines of “data analysis” code to a full software suite.
- Requirements/functionality evolved with the software
- SQA rigor increased as the software grew in scope/complexity

Considerations for other software:

- Software written from the ground up for a specific purpose should consider SQA from the beginning through all stages of development.
- Software written by large teams may have additional procedures

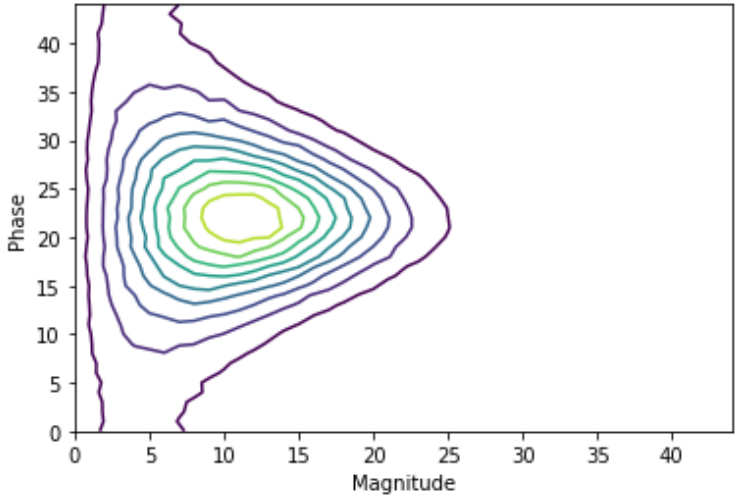
Stage 1: “Help me calculate this uncertainty budget”

- A few lines of code in an interactive Python/Jupyter notebook
- Considered “data analysis”, not really “software”

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
N = 1000000
real = np.random.normal(loc=.01, scale=.01, size=N)
imag = np.random.normal(loc=0, scale=.01, size=N)
magnitude = np.sqrt(real**2 + imag**2)
phase = np.arctan2(imag, real)
```

[3]:
```
counts, ybins, xbins = np.histogram2d(phase, magnitude, bins=45, density=True)
levels = np.linspace(counts.min(), counts.max(), 11)[1:]
plt.contour(counts, levels)
plt.xlabel('Magnitude')
plt.ylabel('Phase');
```



Stage 1: “Help me calculate this uncertainty budget”

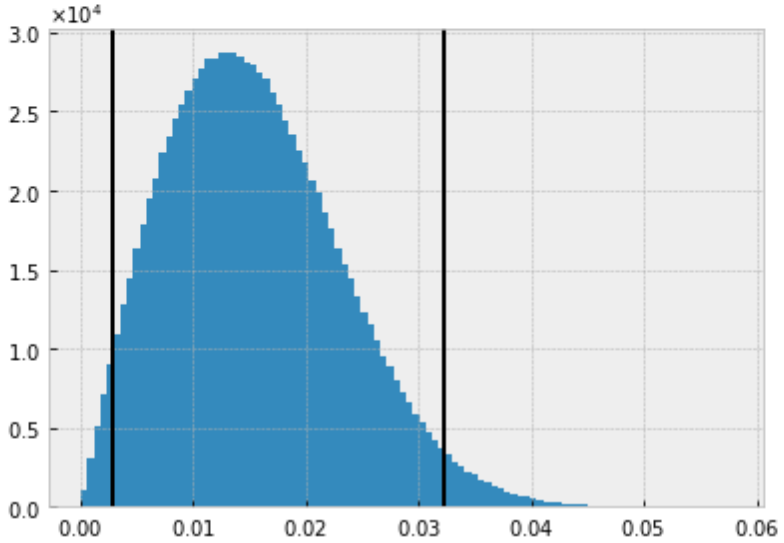
SQA: Low risk, low rigor

No real or formal SQA activities, just a simple sanity check:

- Do the results make sense?
- Did I get a negative uncertainty? Or 100000% uncertainty?

```
[13]: xlow, xhi = np.quantile(magnitude, (0.025, 0.975))
      plt.hist(magnitude, bins=100)
      plt.axvline(xlow, color='black')
      plt.axvline(xhi, color='black')
      print(f'95% Coverage: ({xlow:.3g}, {xhi:.3g})')
```

```
95% Coverage: (0.0029, 0.0323)
```



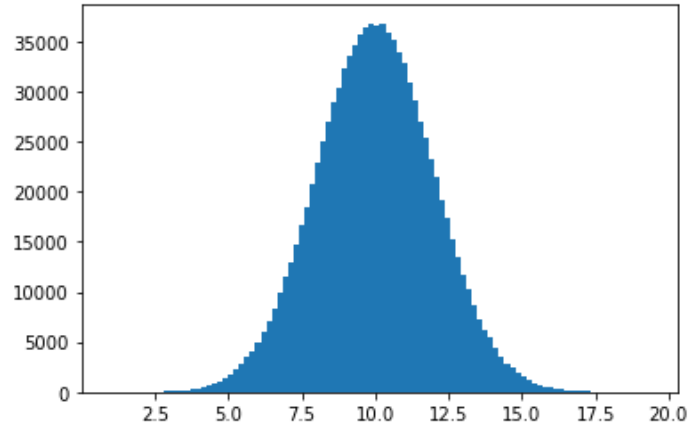
Stage 2: “Now do some other calculations too”

- Calculations were generalized into a few functions
- Still “data analysis”, but with more structure, reusable
- Code shared among a few programmers via email

[16]:
```
def montecarlo_uncert(func, means, uncerts, N=1000000):
    samples = [np.random.normal(m, u, N) for m, u in zip(means, uncerts)]
    results = func(*samples)
    return results
```

[17]:
```
def f(x, y):
    return np.sqrt(x**2 + y**2)

samples = montecarlo_uncert(f, means=(10, .1), uncerts=(2, .1))
plt.hist(samples, bins=100);
```



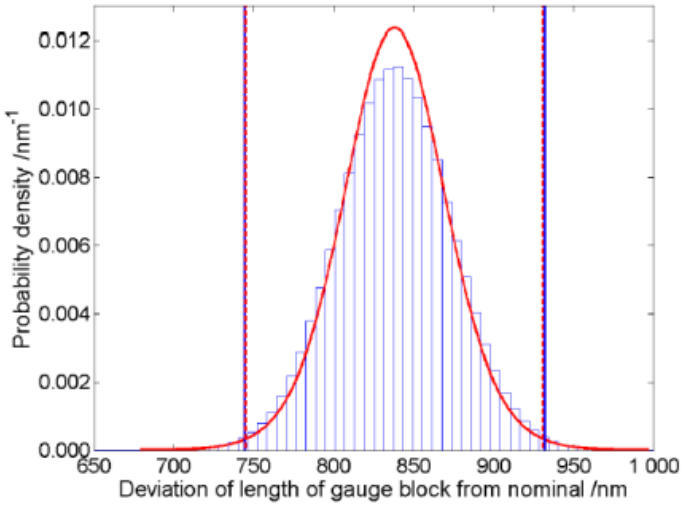
Stage 2: “Now do some other calculations too”

SQA: Low risk, low rigor

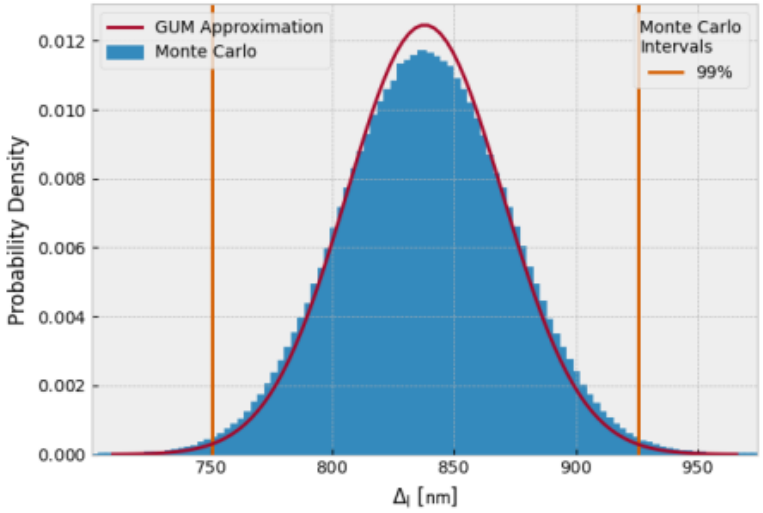
SQA Activities:

- Simple sanity check
- Run a known example (e.g. from NIST-1900) and compare the results
- Revision control: make sure there’s a backup somewhere

GUM Example



Suncal Result



Stage 3: “Hey I can solve these symbolically!”

- Use Sympy library to solve GUM equations symbolically
- Evolved into object-oriented software to handle generalized measurement models with any number of inputs and uncertainty components
- Formed into a proper Python package (installable and importable in Python environment)
- Proper API and more flexible output formatting
- Being used within PSL among several metrologists
- Moving beyond data analysis into something more akin to “software”

Uncertainties:

$$u_{mag} = \sqrt{u_{im}^2 \left(\frac{\partial}{\partial im} mag\right)^2 + u_{re}^2 \left(\frac{\partial}{\partial re} mag\right)^2}$$

$$u_{ph} = \sqrt{u_{im}^2 \left(\frac{\partial}{\partial im} ph\right)^2 + u_{re}^2 \left(\frac{\partial}{\partial re} ph\right)^2}$$

Simplified:

$$u_{mag} = \sqrt{\frac{im^2 u_{im}^2 + re^2 u_{re}^2}{im^2 + re^2}}$$

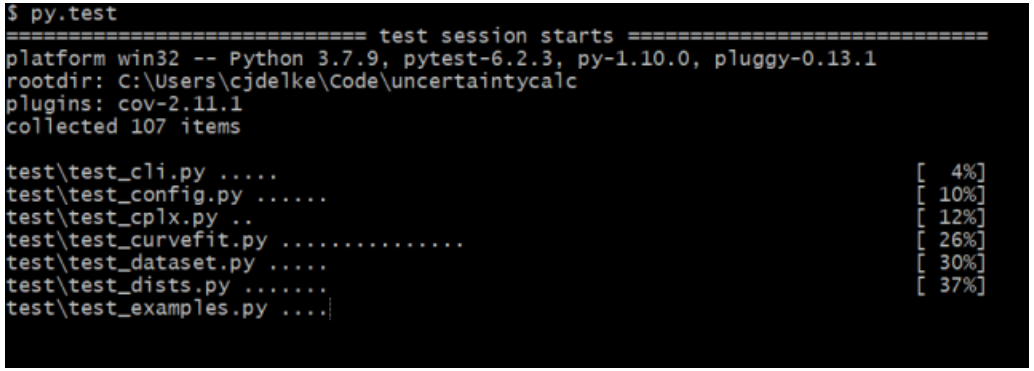
$$u_{ph} = \sqrt{\frac{im^2 u_{re}^2 + re^2 u_{im}^2}{(im^2 + re^2)^2}}$$

Stage 3: “Hey I can solve these symbolically!”

SQA Rigor increases and becomes a documented thing. Higher complexity, potentially more users, but still all internal programmers.

SQA Activities:

- Wrote very high-level software requirements
- Black-box level testing. Use multiple examples from NIST-1900, GUM, etc.
- Formal test cases developed and documented, automated with py.test
- Every requirement traceable to an automated test case
- True revision control in git
- Every user is testing the code, even if they don’t realize it!



Aside: Writing Software Requirements

See ISO/IEC/IEEE 29148:2011 for suggestions on writing quality software requirements:

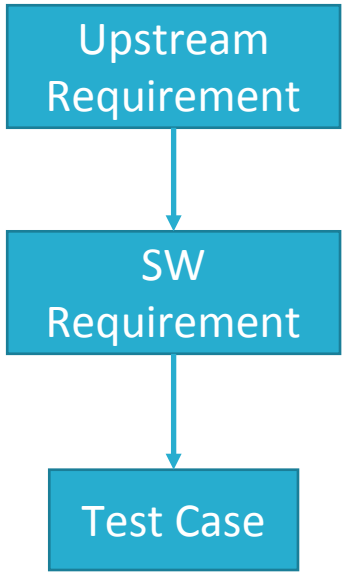
- Necessary
- Implementation-Free
- Unambiguous
- Consistent
- Complete
- Singular
- Feasible
- Traceable
- Verifiable

Poor requirement: “It should calculate uncertainties”

Better requirement: “The software shall compute combined standard uncertainty of a measurement model by applying GUM equation 13 to arbitrary measurement functions…”

Requirements Traceability

Traceability doesn't always refer to measurement values and the SI…



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	**REQUIREMENT**		**TEST CASE**											
2			test_examples.py											
3		Covered by:	test_GUMH2	test_NIST6	test_NIST6_fn	test_NIST10	test_NIST3	test_NISTE11	test_GUMSUP2	test_DEGF	test_GUMH1	test_montecarlo	test_NPLlog	test_XRF
4	**Measurement System - Input**													
5	The software shall:													
6	1. allow entry of one or more measurement model equations (as equation (1) in GUM). Each equation may be independent or depend on other equations in the system. Equations may be entered as:	25	X	X	X	X	X	X	X	X	X	X	X	X
7	a. A string expression	14	X				X	X	X			X	X	X
8	b. A python callable	7			X	X								
9	c. A sympy object	1												
10	2. allow entry of a nominal value for each input variable defined in the measurement equations.	17	X	X	X	X	X	X	X	X	X	X	X	X
11	3. allow entry of one or more uncertainty components for each input variable defined in the measurement equations.	20	X	X	X	X	X	X	X	X	X	X	X	X
12	a. Each uncertainty component shall reduce to a standard uncertainty and degrees of freedom for each input variable for use in the GUM calculation	2									X			X
13	b. Each uncertainty component shall consist of a statistical distribution function and parameters that define that function, for use in the Monte-Carlo calculation. A normal distribution is used by default.	15	X	X	X	X	X	X	X	X	X		X	X

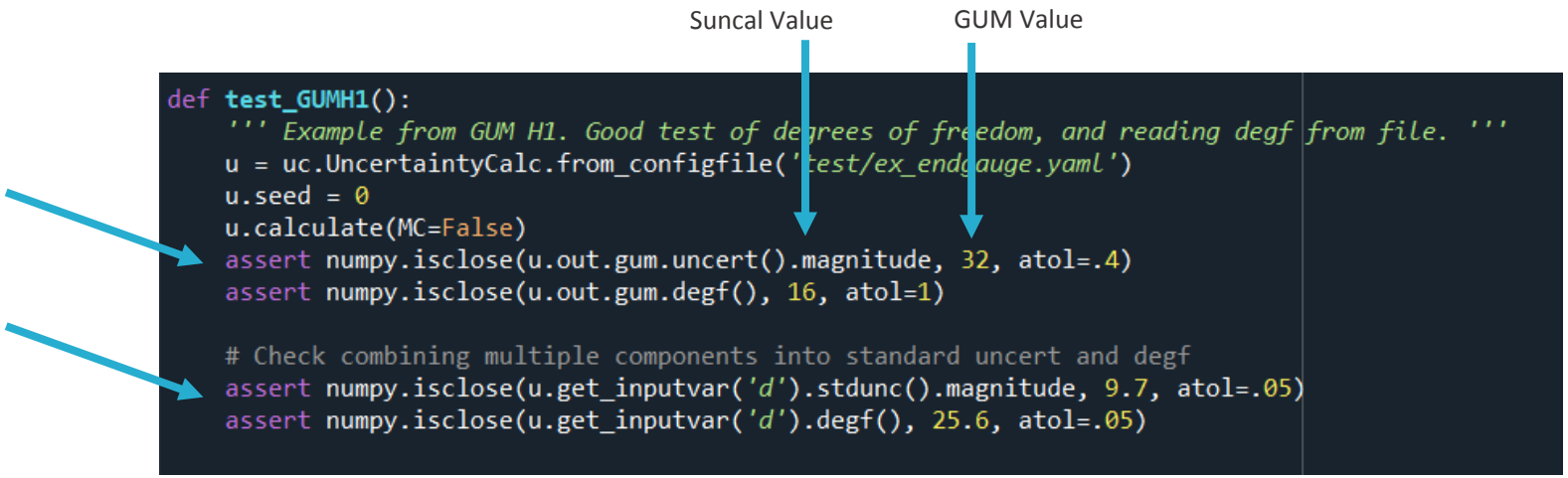
Suncal uses a cheap spreadsheet to record traceability.
Fancier database applications are available for this too.

“Black Box” Test Cases

Individual functions that test one aspect of the code.

Usually run a calculation, then compare the result with known reference value.

“assert” statements throw an error during test if the software’s result is not close* to the published result

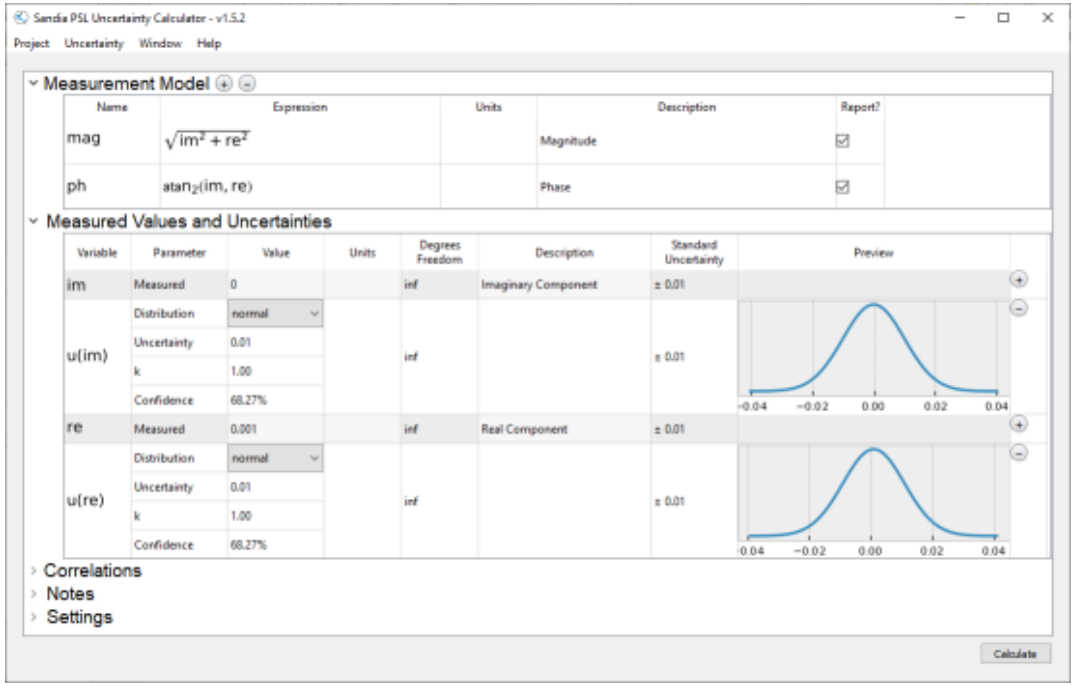


* Within the significant figures reported by the published value

** Be careful when testing Monte Carlo calculations – set the random number seed or results may fail!

Stage 4: “More people would use this if it had a GUI”

- Add full-featured user interface (pyQt) that sits on top of Python library
- Refactored some of library to better integrate with GUI
- Added save/load capability
- Added more integrated output plotting rather than relying on user to generate plots themselves
- Now being used outside PSL at other Sandia orgs, trainings



Aside: Some thoughts on GUIs…

Before you jump in to writing a GUI, ask:

- Do you REALLY need a GUI? Or would a command-line interface be just as good?
- Do you know what OOP means?
- Do you care what language it's programmed in? If not, pick one that makes GUIs easy.

Suncal lines of code:

- 9276 Backend
- 9435 Graphical user interface
- 314 Command-line interface

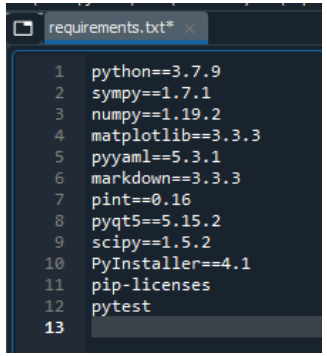
Always start with backend – keep it separate! – then add the GUI on top

Stage 4: “More people would use this if it had a GUI”

SQA Rigor: GUI adds complexity, more rigor needed

SQA Activities:

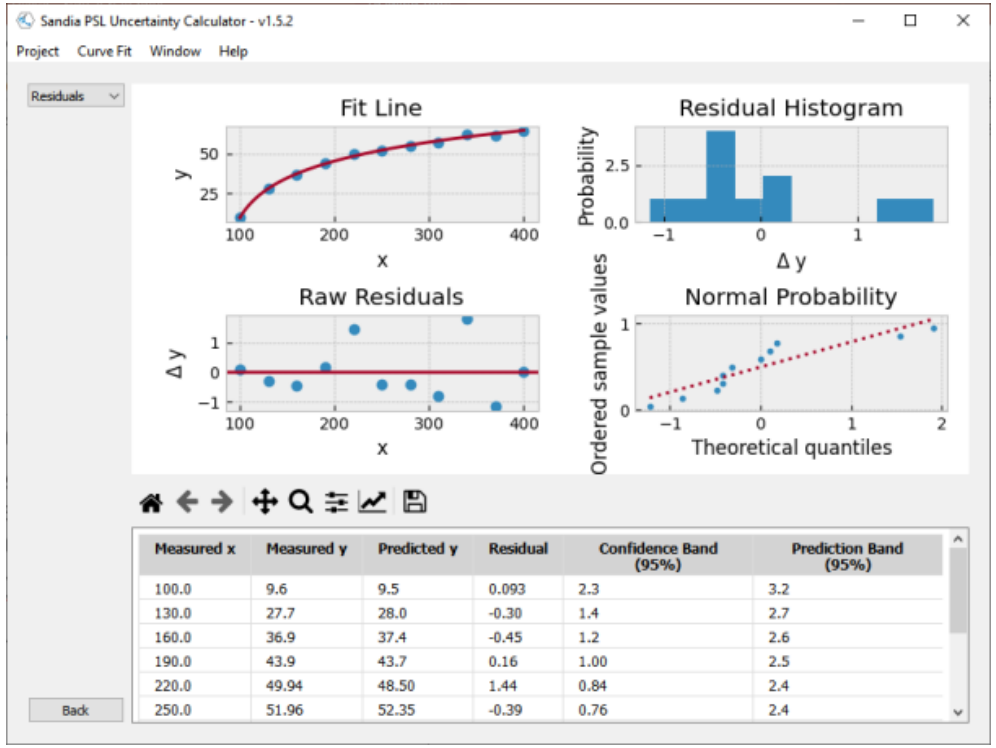
- Add some function-level tests (more than black-box testing)
- GUI needs usability testing. Recruit the interns!
- Add a user manual
- Documentation and configuration management for build process
- Formal issue/change request tracking
- Independent check of SQA activities.



Dependencies locked to specific versions for build/release process

Stage 5: “Let’s share it with the world!”

- Additional calculation features are being added
- Public release means satisfying legal requirements: license file, 3rd party acknowledgements
- Host publicly on Github, add website
- Much expanded user base



Measured x	Measured y	Predicted y	Residual	Confidence Band (95%)	Prediction Band (95%)
100.0	9.6	9.5	0.093	2.3	3.2
130.0	27.7	28.0	-0.30	1.4	2.7
160.0	36.9	37.4	-0.45	1.2	2.6
190.0	43.9	43.7	0.16	1.00	2.5
220.0	49.94	48.50	1.44	0.84	2.4
250.0	51.96	52.35	-0.39	0.76	2.4

Stage 5: “Let’s share it with the world!”

SQA Rigor: Larger audience, higher risk

SQA Activities:

- Regression testing becoming more important
- Added informal code-coverage analysis during testing, leading to additional unit-tests
- Code linting/analysis tools
- Still no formal or automated GUI testing, but more users exercising it

```
62
63 def diagonal(a):
64     ''' Return diagonal of square matrix '''
65     if len(a) > 0 and a[0]:
66         return [a[i][i] for i in range(len(a))]
67     else:
68         return []
69
```

Code-coverage report showing a missed edge case

Current state of Suncal SQA

Suncal is certainly not perfect or bug-free, but we believe SQA activities (still relatively low rigor) are appropriate to ensure calculation results are reliable.

SQA Activities being done:

- Revision control (code, requirements, change requests)
- Informal user testing (via training, etc.)
- Automated functional (black-box) test, with traceability to SW requirements
- Automated unit testing
- Independent review of SQA activities
- Build/configuration management
- Simple code-coverage analysis of automated test to identify additional unit tests
- Code linting/analysis tools

Higher-rigor SQA activities may include:

- Automated GUI test
- Formal usability testing, human-factors assessment
- Python type hinting to help identify/prevent defects
- Formal and documented code reviews
- Code coverage requirements (statement, branch, or condition-level) with justification for non-covered code
- Rigorous change procedure for code and requirements
- Additional independence in SQA activities
- Independent auditing/testing of 3rd party dependencies

Suncal References

Overview of Suncal and its features:

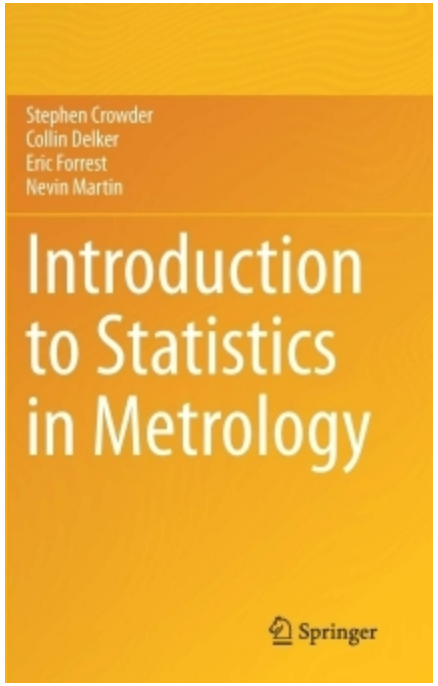
- A comprehensive open-source software for statistical metrology calculations: from uncertainty evaluation to risk analysis. C. Delker. NCSLI Measure 13-3, 2021 (accepted)
- https://sandiapsl.github.io

Hands-on Training:

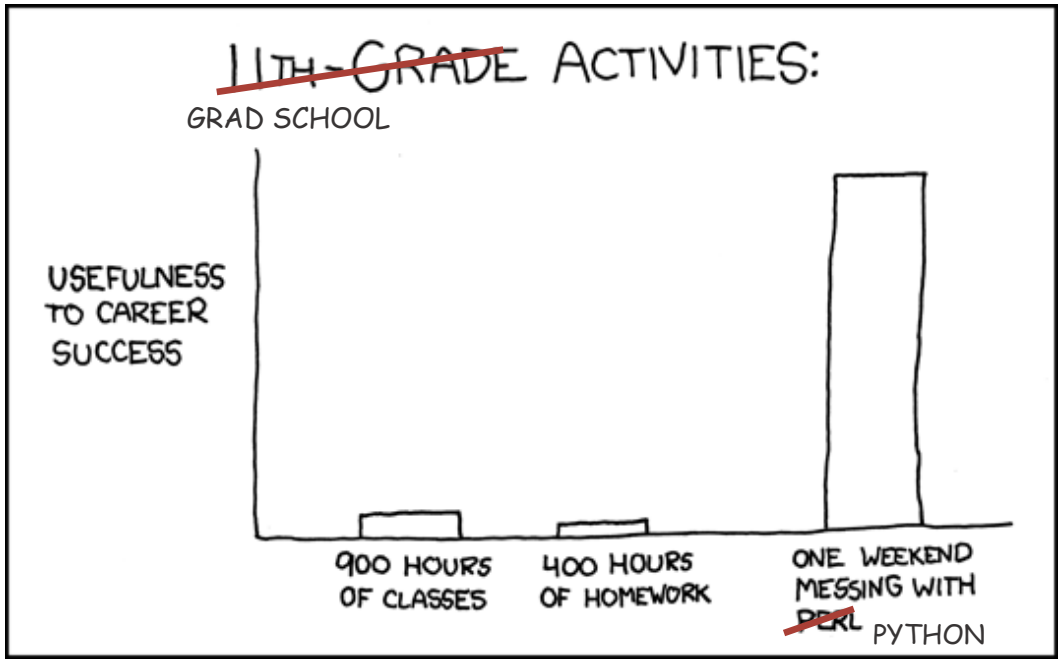
- Introduction to the Suncal Software for Statistical Metrology Calculations. NCSLI Symposium Tutorial Program, 2021.

Theory and statistics behind the calculations:

- Introduction to Statistics in Metrology. S. Crowder, C. Delker, E. Forrest, N. Martin. Springer, 2020.



Questions?



https://xkcd.com/519