Performance Variation of the Albany Land-Ice Code at Sandia National Labs

Kyle Shan

March 11, 2020

Abstract

Albany Land-Ice is a numerical ice sheet model being maintained and developed in part by scientists at Sandia National Labs. With many contributors actively working on the model, improvements in one area may inadvertently affect performance in other ways. In this project, we explore the use of automated changepoint detection methods and visualization tools to detect changes in the runtime of nightly test runs. Generalized likelihood ratio methods are used to evaluate the significance of candidate changepoints. Jupyter notebook and the Plotly package for Python are used to display findings and show interactive plots of multiple test cases. Building off of the changepoint model, we perform paired analysis of different test cases to evaluate one against the other. The resulting notebooks and models are currently being integrated into automated processes and email reports at Sandia. The code for this project is publicly available at `www.github.com/kylecshan/perf-analysis`.

1 Introduction

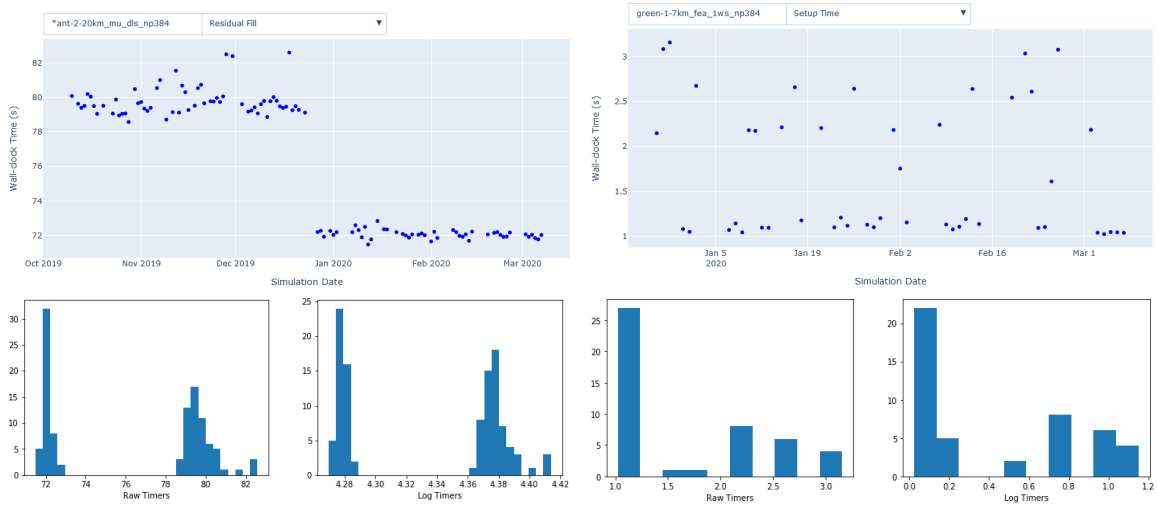
To understand the effects of climate change, accurate and high-performance models must be developed to predict the state of our Earth, looking decades into the future. One of these prediction tasks is the increase in sea level, which is greatly influenced by the behavior of land ice sheets. For this purpose, the Albany Land-Ice model is being developed at Sandia National Labs and Los Alamos National Labs, and is intended to run on various high-performance computing architectures [8]. These models run on large-scale, high-resolution data, so keeping simulation times manageable is of great importance.

The core components of the model are the Albany PDE framework and Trilinos nonlinear solvers. With many scientists actively contributing to both code bases, model improvements in one area can occasionally affect performance in other areas. In particular, this project is concerned with unexpected changes in simulation time, in terms of wall-clock time. Until recently, changes in performance could go unnoticed for weeks or months. To avoid this situation, the current solution at Sandia uses automatically generated plots, which must be reviewed manually; and simple statistical tests, which can detect the presence of changes but not always the point at which they occur.

To improve the performance monitoring process, we explore using changepoint detection models and creating automatically-generated visualizations to:

- Measure the likelihood of a change in performance
- Detect the most likely location of a change, and relate it to Git commit metadata
- Show performance trends over time

With a changepoint model in place, we can also identify which data is relevant to present-day decisions. For example, we can compare times from test cases run on different computing architectures, while excluding times recorded with an outdated code base that are no longer relevant.



(a) (b)

Figure 1: Time series plot and histogram of various timers, as either the raw data (subplot bottom left) or log-transformed data (bottom right).

In section 2, we discuss the dataset of test cases and timers. In section 3, we cover methods for changepoint detection and visualization libraries and show some examples. In section 4, we discuss the performance of our model. Lastly, in section 5 we recap the project experience, and discuss potential improvements for the future.

2 Data and Processing

The data are provided by Sandia. When nightly performance tests are run, timers and metadata are collected with ctest and saved in `.json` files, starting in October 2019. Test are currently being done on the Blake and Waterman (since Jan 2020) test beds [14]. Figures A.1 and A.2 show the various test cases, and Figure A.3 shows the timers that are collected during the modeling process. We rarely have known instances of performance changes, so our evaluation of detection methods relies mostly on human judgment and simulated data.

The lack of known changes also makes it difficult to test the distribution of the data, since each time series could contain data from multiple distributions (i.e. different code versions with different performance characteristics). There are many combinations of test cases and timers, making it prohibitive to visually inspect each one. Based on a quick sampling, we conclude that the normal or lognormal distribution describes most of our data reasonably well, except for occasional outliers in the positive direction. In Figure 1, we show two cases where any distribution changes are fairly visible. In the first case, after splitting the data, the values and log-transformed values both pass the D'Agostino and Pearson normality test [4] (using the implementation of `scipy.stats.normaltest`), with the log-transformed data fitting slightly better. In the second case, the data do not seem to fit any standard distribution.

Another observation from inspecting the data is that changes may happen with very few (as little as three) observations in between. Hence, our methods cannot rely on having even a moderate amount of data.

Algorithm 1 The single changepoint detection algorithm. Parameters are the significance level, α, and the number of large changes to consider, k. Input is assumed to be roughly normal. Returns a set C of viable changepoint candidates, along with a map S from the candidates to their t-statistics.

function SINGLECHANGEPOINT($x_{1:n}, \alpha, k$)
  $C = []$
  $S = \{\}$
  $t^* = t_{n-2}^{1-\alpha/(2\max(n,k))}$
  $\mathcal{K} = \text{ARGSORTDESC}(|x_{2:n} - x_{1:n-1}|)$
  for $\nu \in \mathcal{K}_{1:\max(n,k)}$ **do**
    $t = \text{T-TEST}(x_{1:\nu-1}, x_{\nu:n})$
    if $|t| > t^*$ **then**
      $C \leftarrow [C, \nu]$
      $S \leftarrow S \cup \{\nu \to t\}$
    end if
  end for
  return C, S
end function

3 Methods

The task of identifying changes in performance falls in the category of problems known as changepoint detection; see [1], [13], [3] for a complete description of this type of problem.

3.1 Single Changepoint Detection

The basic building block of changepoint detection algorithms is single changepoint detection: given a univariate time series $x_1, x_2, \ldots, x_n$ such that $\{x_i\}_{i=1}^{\nu-1}$ are i.i.d. with distribution f_0, and $\{x_i\}_{i=\nu}^{n}$ are i.i.d. with distribution f_1; what is the best estimate of ν? The conventional approach assumes a known baseline distribution f_0 (and sometimes f_1 as well).

Since we have many separate time series, each potentially changing multiple times, we need a method which uses limited knowledge of f_0 and f_1. In the most general case, nonparametric models can theoretically detect any change in distribution, but they can be less powerful than parametric models. Since our sample size is limited, we assume a lognormal distribution, based on our findings in Section 2.

Given a potential changepoint ν, we can perform a two-sample t-test of $\{x_i\}_{i=1}^{\nu-1}$ and $\{x_i\}_{i=\nu}^{n}$. We found that Student's t-test, assuming equal variances, worked reasonably well, although a future improvement could be to use Welch's t-test which is robust to unequal variance.

To find the most likely changepoint, we can apply the t-test to each candidate changepoint. However, we must adjust our desired significance level for multiple hypothesis testing. The simplest solution is the Bonferroni correction [2], i.e. testing each candidate at a significance level of $\alpha/(n-1)$, where α is our original desired significance level. Figure 2 shows an example of this test. This correction is known to be overly conservative for large numbers of tests. To alleviate this, we test only the $\min(k, n-1)$ largest changes (by absolute value) in the time series; a typical value of k could be between 3 and 10. Lower values of k allow for more powerful tests, but are more susceptible to outliers clogging the list of the largest changes. Algorithm 1 shows pseudocode for this building block algorithm.

The multiple t-test method above can also be viewed as a generalized likelihood ratio test. Define two families of hypotheses: the null hypothesis states that $\{x_i\}_{i=1}^{n}$ belongs to a single distribution, while the alternative states that there exists some $\nu \in \{2, 3, \ldots, n\}$ such that $\{x_i\}_{i=1}^{\nu-1}$ and $\{x_i\}_{\nu}^{n}$ are separate distributions. The test statistic is found by maximizing the likelihood of each hypothesis and

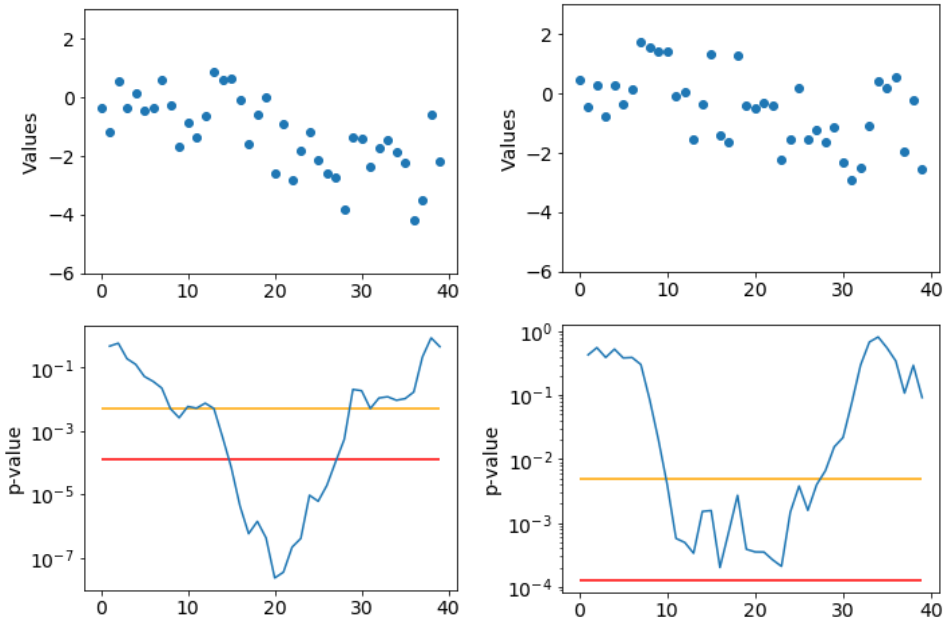


Figure 2: Simulated data (top) with the p-values resulting from our generalized likelihood ratio test (bottom). Left: the p-value for a t-test is minimized when splitting the data in the middle, passing both our original significance level of 0.5% and the Bonferroni-corrected significance level of $0.5\%/(40-1)$. Right: a more difficult case; none of the candidate changepoints fall below the corrected significance level.

taking the likelihood ratio. Assuming equal variance, Student's t-test is equivalent to the likelihood ratio for a given candidate ν [6].

Besides evaluating a change in the mean, as in [10] [12] [6], it is also possible to test for a change in variance [7]; however, this additional testing appeared to be unhelpful for our application.

3.2 Multiple Changepoint Detection

Our time series data is likely to have multiple changepoints. A popular approach to multiple changepoint detection is to use a greedy algorithm which splits the data recursively until no more significant changepoints can be found. This approach is called binary segmentation [11], and there are various extensions to make the process more robust, such as wild binary segmentation [5]. We instead use a simpler sequential method, where we emulate gathering one new observation at a time. If a changepoint is detected, we mark it as such, backtrack to immediately after the changepoint, and future analysis will disregard any data prior to that changepoint. While this process may not find the optimal changepoints, it is guaranteed not to retroactively change its decisions, which is easier to understand. Figure 3 shows an example of our method applied to simulated data, and Algorithm 2 provides pseudocode.

3.3 Robustness to Outliers

Outliers present a significant hurdle to the task of changepoint detection. Almost by definition, an outlier comes from a different distribution than the rest of the data, so changepoint models are understandably keen on detecting them. We take several steps to mitigate this effect while minimally decreasing the power of our tests.

First, in any single t-test, we first examine the data on either side of the candidate changepoint for outliers using the median absolute deviation [9], with a threshold comparable to three standard deviations. We remove data beyond this threshold, up to a tenth of the total data.

Second, we require three consecutive detections of the same changepoint before marking it as such. The reasoning here is that the influence of the outlier will be diluted by a couple inliers following it, which will make the change appear less significant. However, consecutive outliers are still a problem for this technique.

Lastly, we limit the lookback window of our method to 30 observations. A common critique of p-values is that the smallest of changes becomes significant when samples sizes are large. Setting a maximum window size helps to avoid hypersensitivity as we get further from the last detected changepoint.

Algorithm 3 describes our final changepoint detection model. To illustrate the effect of these modifications, Figure 4 shows how occasional outliers can trigger false detections without our adjustments, and that our adjustments allow the model to ignore these outliers.

3.4 Automated Dashboard Report

To facilitate the review of performance changes, we build a Jupyter notebook which loads `.json` files from a specified directory and runs our changepoint model. The mean and standard deviation is calculated between changepoints; this information is plotted along with the time series. Using the Plotly package for Python, we create interactive drop-down menus which allow selecting the test case and timer. This package also has hover-text functionality, which we use to show Git commit information. With this relatively small number of interactive features, we can export the notebook to an offline HTML document, which can be viewed on most browsers. See Figure 5 for an example of this report.

We also generate a similar notebook for comparing test cases. In addition to the time series plot, we also perform paired analysis by taking the intersection of dates where both test cases ran successfully. We find that in general, idiosyncrasies in cluster performance, as well as some performance changes,

Algorithm 2 The multiple changepoint detection algorithm. Parameters are the same as Algorithm 1. Returns a set C of changepoints.

function MULTIPLECHANGEPOINT($x_{1:n}, \alpha, k$)
  $\mathcal{C} = []$
  $i = 0$
  $j = i + 1$
  while $j \leq n$ **do**
    $C_j, S_j \leftarrow$ SINGLECHANGEPOINT($x_{i:j}, \alpha, k$)
    if $C_j \neq \emptyset$ **then**
      $\nu^* = \arg\max_{\nu \in C_j} S_j(\nu)$
      $\mathcal{C} \leftarrow [\mathcal{C}, \nu^*]$
      $i \leftarrow \nu^*$
      $j \leftarrow \nu^* + 1$
    else
      $j \leftarrow j + 1$
    end if
  end while
  return $\mathcal{C}$
end function

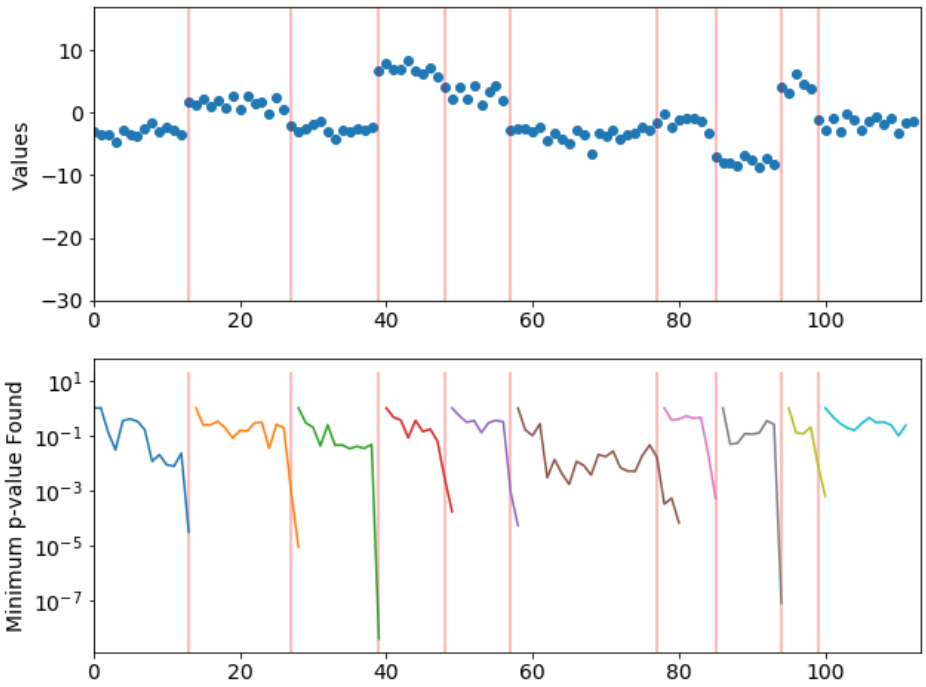


Figure 3: A multiple changepoint detection problem (top), and the p-values at the moments of detection (bottom).

Algorithm 3 The robust multiple changepoint detection algorithm. In addition to the parameters of Algorithm 2, we also define the minimum number of consecutive detections, m, and the maximum lookback window, w. We call a modified version of the single changepoint algorithm, SingleChangepointRemoveOutliers, which removes outliers from each sample prior to performing a t-test. Returns a set C of changepoints.

function RobustMultipleChangepoint($x_{1:n}, \alpha, k, m, w$)
  $\mathcal{C} = []$
  $i = 0$
  $j = i + 1$
  while $j \leq n$ **do**
    $C_j, S_j \leftarrow$ SingleChangepointRemoveOutliers($x_{i:j}, \alpha, k$)
    $\mathcal{V} = C_{j-m+1} \cap C_{j-m+2} \cap \ldots \cap C_j$
    if $\mathcal{V} \neq \emptyset$ **then**
      $\nu^* = \arg\max_{\nu \in \mathcal{V}} \left(\sum_{\ell=j-m+1}^{j} S_\ell(\nu) \right)$
      $\mathcal{C} \leftarrow [\mathcal{C}, \nu^*]$
      $i \leftarrow \nu^*$
      $j \leftarrow \nu^* + 1$
    else
      $j \leftarrow j + 1$
      $i \leftarrow \max(i, j - w)$
    end if
  end while
  return $\mathcal{C}$
end function

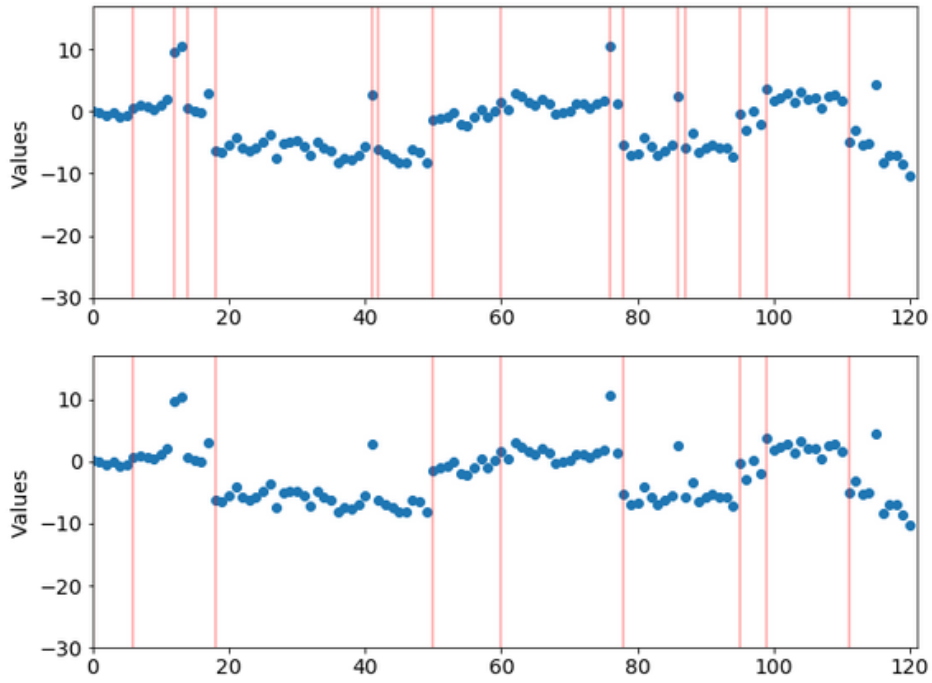


Figure 4: Our multiple changepoint detection algorithm without (top) and with (bottom) the tweaks for robustness against outliers.

Performance Timelines

```
Events in the most recent 10 days:
02/26/2020:
    ant-2-20km_mu_dls_np384: Total Time
                             NOX Total Preconditioner Construction
                             NOX Total Linear Solve
    green-1-7km_fea_1ws_np8: Residual Fill Evaluate
02/27/2020:
    green-1-7km_fea_mem_np8: Residual Fill
02/29/2020:
    green-1-7km_fea_1ws_np8: Jacobian Fill Export
    green-1-7km_fea_mem_np8: Total Time
                             Total Fill Time
                             Jacobian Fill
                             NOX Total Linear Solve
03/03/2020:
    ant-2-20km_mu_dls_np384: Jacobian Fill
```

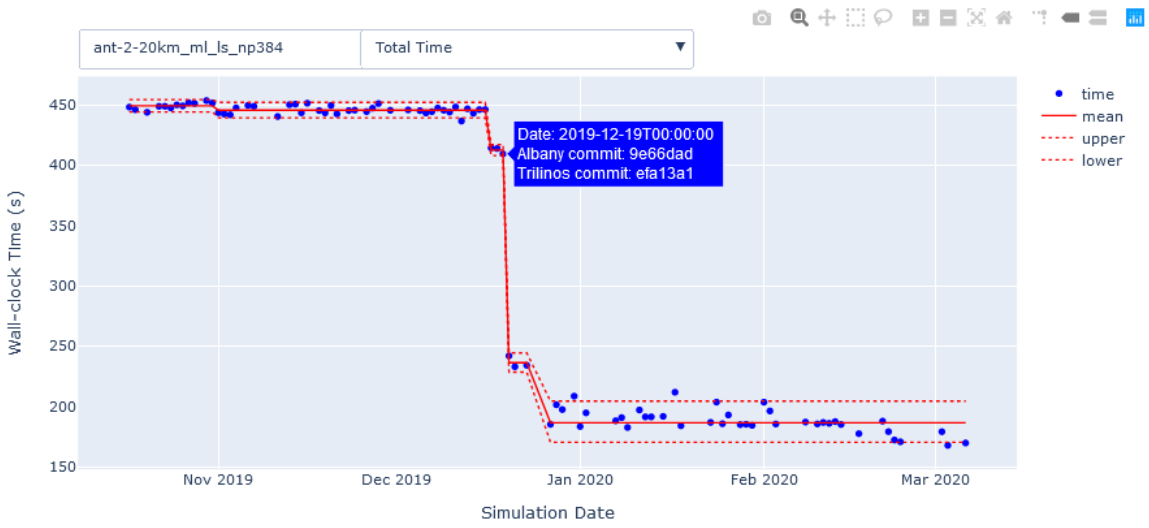


Figure 5: An example of the offline HTML report, with interactive drop-down menus and hover text.

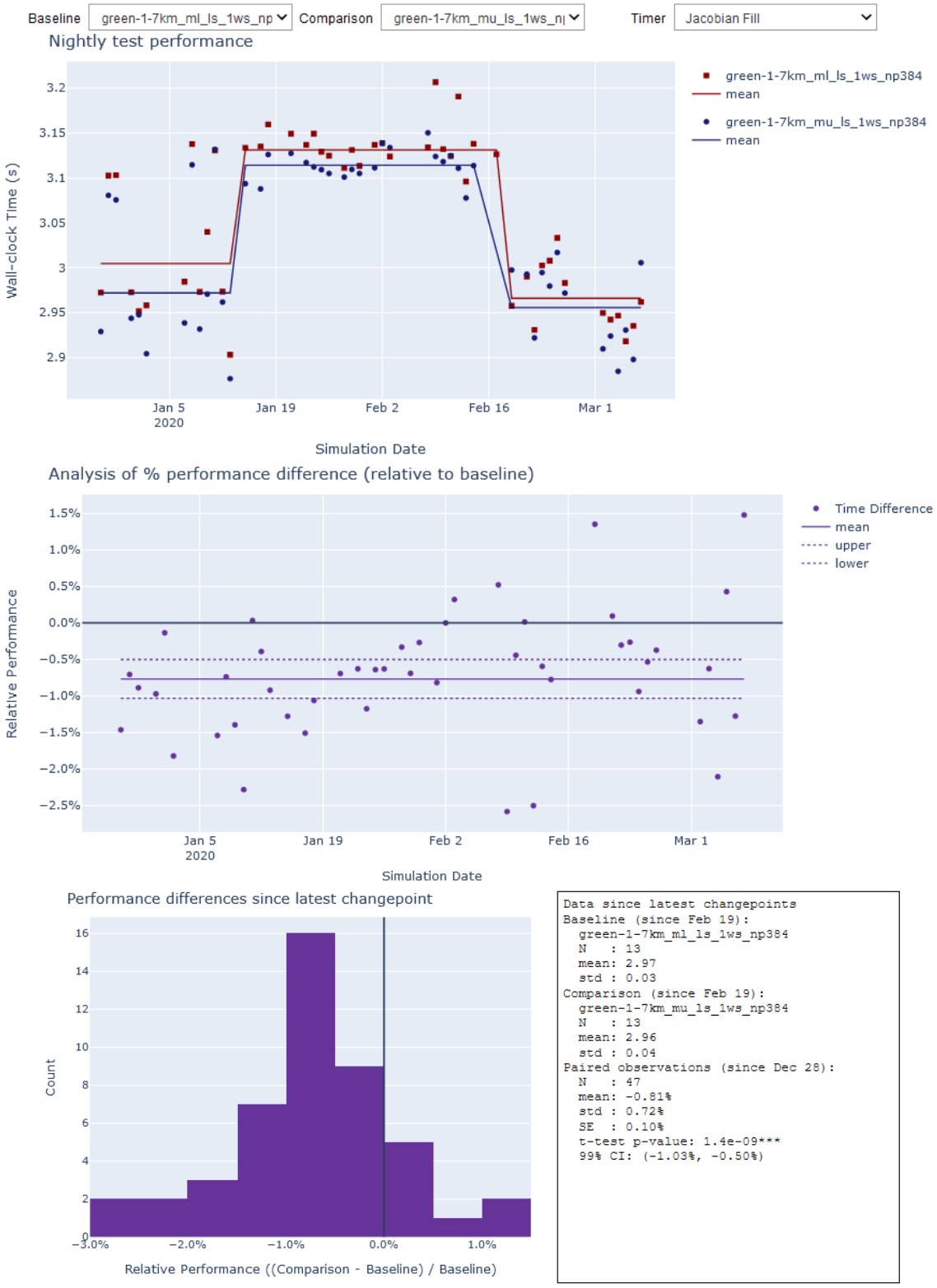


Figure 6: An example of the comparison notebook, with paired analysis of the two test cases.

are highly correlated between different test cases, and a paired analysis removes some of the variation when trying to compare two time series.

For our paired analysis, we take the difference between the log-transformed timers for two test cases, and compute the exponent minus one, yielding a relative performance difference between one test case (the Baseline) and another (the Comparison). We use our changepoint analysis on the log data, and show the mean between changepoints and a 99% confidence interval for the mean. Furthermore, we include a histogram of the relative differences since the last changepoint, and summary statistics including the significance of a t-test for the mean relative difference. Due to the additional computations that require a Python backend, this notebook must have a Python kernel to use the drop-down menus, and the interactivity cannot be used in an exported HTML file. See Figure 6 for an example of this report.

4 Results

Since we have very few cases of confirmed performance changes, our performance analysis relies mostly on human judgment. In Section 3, we showcased our model on simulated examples, which generally worked well. When used on actual data, the detection of obvious changes is good, but outliers still present occasional difficulties. This is best shown by example; see Figure 7 below. In the top graph, two consecutive outliers are mistakenly identified as changepoints, but the impact on our conclusions is minimal. In the middle graph, the data contains several large outliers in every regime; an ideal model would ignore these outliers and focus on the small range of variation when they are excluded. In the bottom graph, we found a mysterious upward trend in runtime. The changepoint model detects this trend, but the exact locations of the changepoints are essentially arbitrary.

5 Conclusion and Future Work

In this project, we implemented a relatively simple changepoint detection method, with several modifications to improve robustness with our dataset. We wish we could have explored changepoint detection algorithms more thoroughly, but there are few implementations available online, and we had to pick a course for the ten-week quarter. The simple algorithm is, at least, easily understood, and the results are generally good.

Our Jupyter notebooks showcase the changepoint model, and are a proof of concept for automatically-generated, interactive reports. Although we were unfamiliar with interactive tools in Jupyter notebook, we were able to achieve our target functionality, although the comparison notebook is not as portable as initially hoped. We are pleased to have received positive feedback on these notebooks, which should be an upgrade to the automatically-generated notebooks currently being used at Sandia. Our changepoint model is also being integrated into email summaries of nightly test data, and there is the potential for other teams at Sandia to utilize these tools too.

The code developed in this project is available at `www.github.com/kylecshan/perf-analysis`.

5.1 Future Work

We identify a few areas for improvement in both the changepoint model and the visualization tools.

Our changepoint model does not always make the judgements we expect, particularly when outliers are involved. While we tried a few modifications to reduce the effect of outliers, there are other options that could be explored. We noted previously that an outlier is effectively a changepoint, but not one we're typically interested in. We could test whether the data have reverted to a previous distribution; if so, we would mark the interjecting points as outliers, and resume as if they did not exist. Hence we acknowledge the existence of outliers, while not letting them affect future decisions.

Another option would be to run test cases multiple times a day. This would allow the effects of outliers to be diluted by non-outliers. However, the current code would need to be modified to

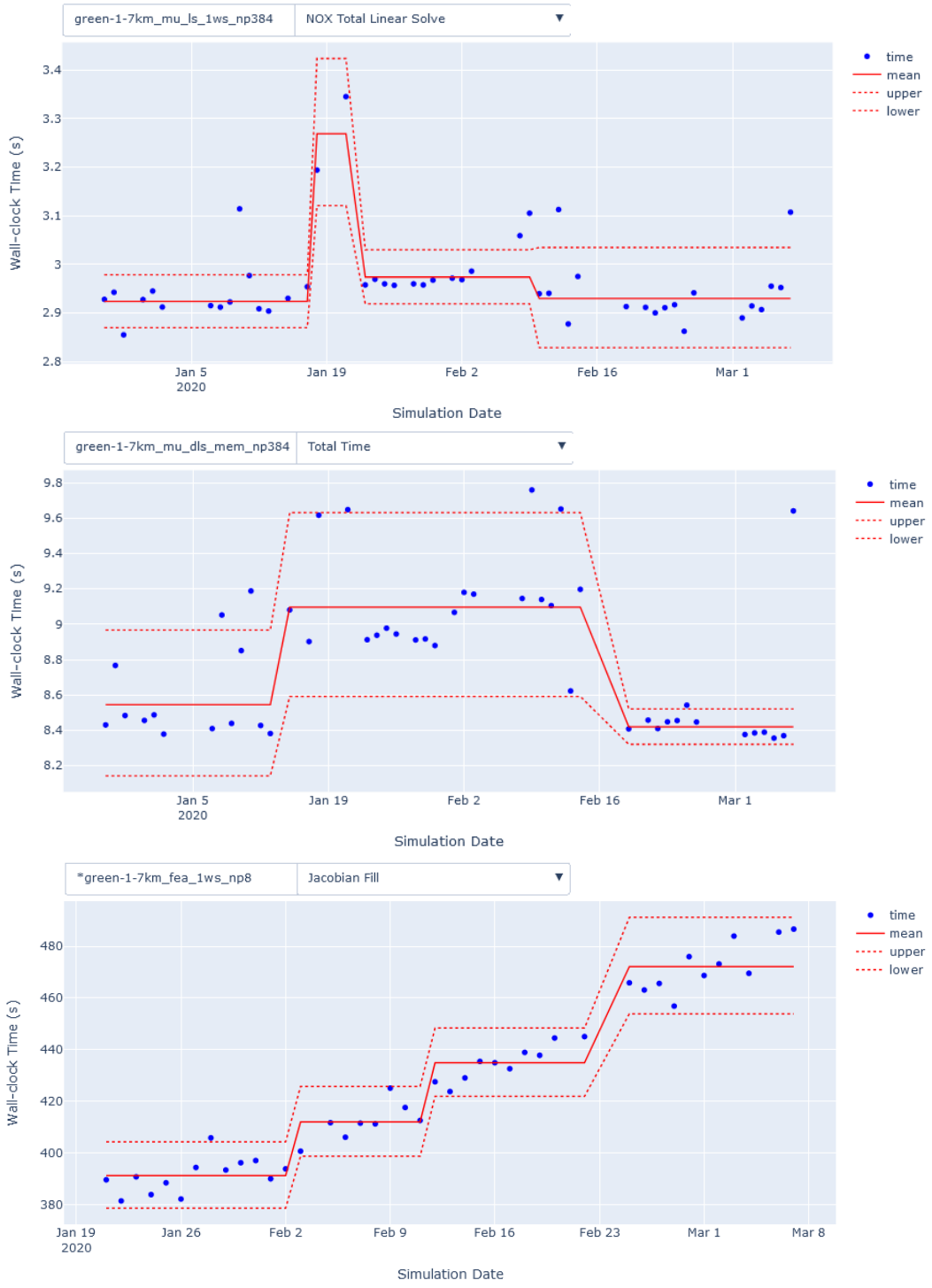


Figure 7: Several examples of the changepoint model's output for actual timer data.

recognize that changepoints can only occur between days. In fact, this logic could be extended to only select changepoints between code commits, since a couple days might occasionally pass between commits.

As mentioned in Section 3, using Welch's t-test may also yield better results, although some brief testing did not show it helping with outliers. Other options to improve the model include using more sophisticated methods, such as nonparametric methods. Given the almost-normality of most of our data, this seemed less likely to help, particularly within the constraint of a ten-week quarter.

In creating the notebooks, we focused on providing useful information, but as a result they could use some additional polish. A server-hosted Python app would leave the most room for improvement, but with the cost of hosting the app. For example, while the single-test-case notebook can be exported to HTML, two drop-downs is essentially the limit of this method. With the recent popularity of data visualizations, there exist many packages focused on enabling (relatively) novice programmers to create dashboards with a wide range of interaction and plotting options, which can be viewed in both desktop and mobile browsers.

References

[1] Samaneh Aminikhanghahi and Diane J Cook. A survey of methods for time series change point detection. *Knowledge and information systems*, 51(2):339–367, 2017.

[2] Carlo E Bonferroni, C Bonferroni, and CE Bonferroni. Teoria statistica delle classi e calcolo delle probabilita'. 1936.

[3] Boris Brodsky. *Change-point analysis in nonstationary stochastic models*. CRC Press, 2016.

[4] RALPH D'AGOSTINO and Egon S Pearson. Tests for departure from normality. *Biometrika*, 60(3):613–622, 1973.

[5] Piotr Fryzlewicz et al. Wild binary segmentation for multiple change-point detection. *The Annals of Statistics*, 42(6):2243–2281, 2014.

[6] Douglas M Hawkins, Peihua Qiu, and Chang Wook Kang. The changepoint model for statistical process control. *Journal of quality technology*, 35(4):355–366, 2003.

[7] Douglas M Hawkins and KD Zamba. Statistical process control for shifts in mean or variance using a changepoint formulation. *Technometrics*, 47(2):164–173, 2005.

[8] M. J. Hoffman, M. Perego, S. F. Price, W. H. Lipscomb, T. Zhang, D. Jacobsen, I. Tezaur, A. G. Salinger, R. Tuminaro, and L. Bertagna. Mpas-albany land ice (mali): a variable-resolution ice sheet model for earth system modeling using voronoi grids. *Geoscientific Model Development*, 11(9):3747–3780, 2018.

[9] Christophe Leys, Christophe Ley, Olivier Klein, Philippe Bernard, and Laurent Licata. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology*, 49(4):764–766, 2013.

[10] Moshe Pollak, David Siegmund, et al. Sequential detection of a change in a normal mean when the initial value is unknown. *The Annals of Statistics*, 19(1):394–416, 1991.

[11] Ashish Sen and Muni S Srivastava. On tests for detecting change in mean. *The Annals of statistics*, pages 98–108, 1975.

[12] David Siegmund and ES Venkatraman. Using the generalized likelihood ratio statistic for sequential detection of a change-point. *The Annals of Statistics*, pages 255–271, 1995.

[13] Alexander Tartakovsky, Igor Nikiforov, and Michele Basseville. *Sequential analysis: Hypothesis testing and changepoint detection*. CRC Press, 2014.

[14] www.sandia.gov. Advanced system technology test beds.

Appendix

Case Name	Processes	Description
ant-2-20km_ml_ls	384	Unstructured 2-20km AIS, ML w/ line smoothing
ant-2-20km_mu_ls	384	Unstructured 2-20km AIS, MueLu w/ line smoothing
ant-2-20km_mu_dls	384	Unstructured 2-20km AIS, MueLu w/ decoupled line smoothing
greenland-1-7km_fea_1ws	384	Unstructured 1-7km GIS, finite element assembly only, single workset
greenland-1-7km_ml_ls_1ws	384	Unstructured 1-7km GIS, ML w/ line smoothing, single workset
greenland-1-7km_mu_ls_1ws	384	Unstructured 1-7km GIS, MueLu w/ line smoothing, single workset
greenland-1-7km_mu_dls_1ws	384	Unstructured 1-7km GIS, MueLu w/ decoupled line smoothing, single workset
greenland-1-7km_fea_mem	384	Unstructured 1-7km GIS, finite element assembly only, memoization
greenland-1-7km_ml_ls_mem	384	Unstructured 1-7km GIS, ML w/ line smoothing, memoization
greenland-1-7km_mu_ls_mem	384	Unstructured 1-7km GIS, MueLu w/ line smoothing, memoization
greenland-1-7km_mu_dls_mem	384	Unstructured 1-7km GIS, MueLu w/ decoupled line smoothing, memoization

Figure A.1: Test cases on Blake test bed

Case Name	Processes	Description
greenland-1-7km_fea_1ws	8	Unstructured 1-7km GIS, finite element assembly only, single workset
greenland-1-7km_fea_mem	8	Unstructured 1-7km GIS, finite element assembly only, memoization

Figure A.2: Test cases on Waterman test bed

Timer Name	Level	Description
Albany Total Time	0	Total wall-clock time of simulation
Albany: Setup Time	1	Preprocess
Albany: Total Fill Time	1	Finite element assembly
Albany Fill: Residual	2	Residual assembly
Albany Residual Fill: Evaluate	3	Compute the residual, local/global assembly
Albany Residual Fill: Export	3	Update global residual across MPI ranks
Albany Fill: Jacobian	2	Jacobian assembly
Albany Jacobian Fill: Evaluate	3	Compute the Jacobian, local/global assembly
Albany Jacobian Fill: Export	3	Update global Jacobian across MPI ranks
NOX Total Preconditioner Construction	1	Construct Preconditioner
NOX Total Linear Solve	1	Linear Solve

Figure A.3: Timer hierarchy for nightly test runs