TBW Financial Model Code Review

David Gorelick and David Gold

June 2020

TBW Financial Model

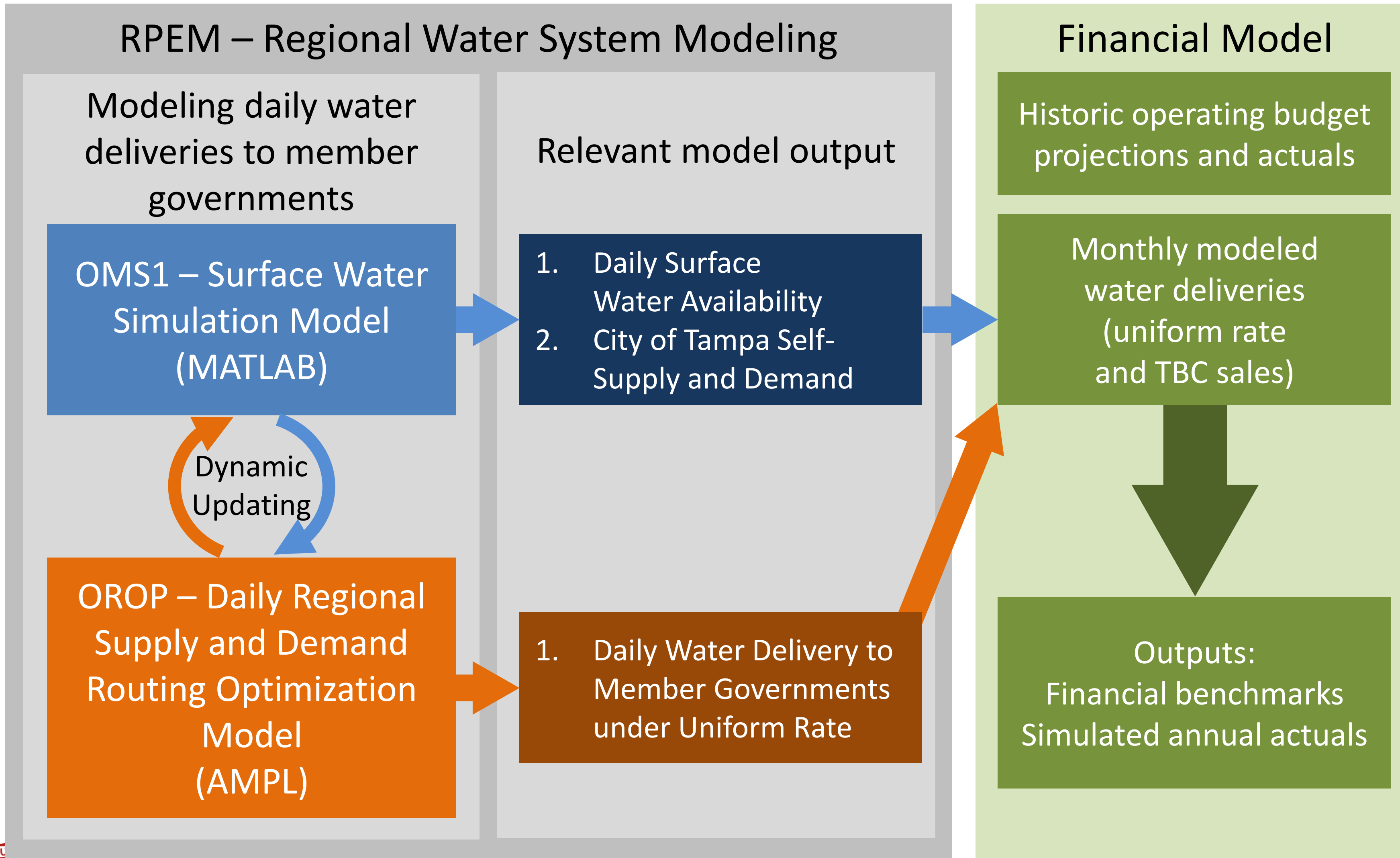
Goal: explicitly model financial impacts of

A. Future changes to water supply system infrastructure and operations (and baseline state)
B. Changes in demand and hydrologic conditions
C. Management intervention (e.g. maintaining a low uniform rate)

Outcomes: measure annual performance based on

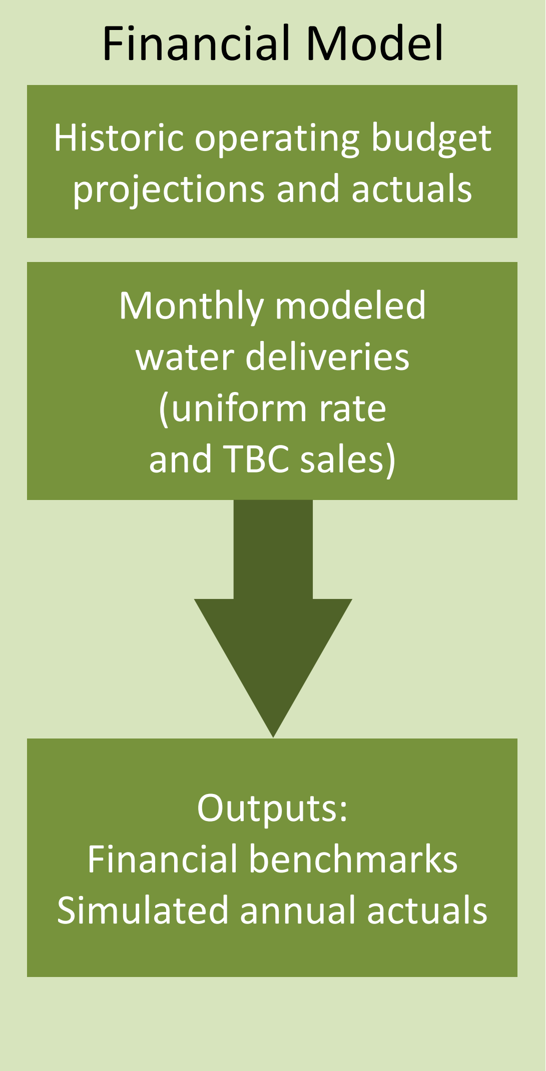
A. Debt and rate covenants (coverage ratios)
B. Reserve fund balances
C. Additional factors of interest?

Financial Model in Context



TBW Financial Model - Breakdown

1. Historic financial information (many thanks to Sandro Svrdlin for his help locating and explaining data)
    - Fiscal Year (FY) actuals
    - Approved annual operating budgets (and future projected budgets)
    - Reserve fund balances, additional details
2. OROP/OMS simulated water demand and deliveries from 2020-2040
    - By member government, aggregated monthly
3. Financial model simulation
    - Monthly water sales/revenues
    - Annual actual outcomes
    - Annual projection of next-year budget and rate-setting
4. Outputs for decision-makers



Code Structure for 1 Realization

OROP/OMS: 1,000 realizations of 2020-2040 water deliveries to member governments
(Monte Carlo of demand and hydrologic conditions)

↘ **1 realization (2020-2040)**

↘ **Enter realization function** (run_FinancialModelForSingleRealization)

- Set parameters, constants, other variables (see below slide for detail)
- Read in historical FY water sales and budgeted/actual financial data
  (build_HistoricalMonthlyWaterDeliveriesAndSalesData, build_HistoricalAnnualData, build_HistoricalProjectedAnnualBudgets, append_Late2019DeliveryAndSalesData)
- Read in water supply modeling realization data
  (read_AMPL_csv, read_AMPL_out, get_HarneyAugmentationFromOMS)
  (if water supply modeling has been modified to note when new infrastructure is triggered, include this factor)
- Read in existing debt/issued bonds and potential future project costs
  (get_ExistingDebt, get_PotentialInfrastructureProjects)

↘ **Enter annual loop from Jan 2020 – Dec 2039**

↘ **Enter monthly loop Jan – Dec each calendar year**

- Calculate monthly water sales revenues
  (get_MemberDeliveries)
  - Use variable uniform rate, TBC rate, and budgeted estimate of fixed monthly payments for each member, as well as past or current FY water deliveries to calculate sales revenue
  - Append to historical record the “observed” deliveries/sales

Code Structure for 1 Realization – Monthly

↘ …

↘ **Enter monthly loop Jan – Dec each calendar year**

- Calculate monthly water sales revenues
  (get_MemberDeliveries)
  - Use variable uniform rate, TBC rate, and budgeted estimate of fixed monthly payments for each member, as well as past or current FY water deliveries to calculate sales revenue
  - Append to historical record the "observed" deliveries/sales
- Check if new infrastructure projects are triggered in water supply modeling
  (check_ForTriggeredProjects)
  - If so, record the ID as a new project to finance, referencing IDs read in by get_PotentialInfrastructureProjects

↘ **If September (end of FY), record actuals and estimate upcoming FY budget**

1. Collect current FY monthly water revenue for all months
2. Pull current FY actuals and budgeted estimates
   (debt service, acquisition credits, unencumbered funds from previous FY, budgeted gross revenues, fixed and variable operational expenses, non-sales revenues, transfers in as revenue from Rate Stabilization, R&R, and Other Funds)

Actuals equal to "observations" from Water Supply Modeling

Actuals assumed equal to Budgeted Amounts

Actuals differ from Budgeted Amounts by Deeply Uncertain Factor

Code Structure for 1 Realization – End of FY

- …
  - **If September (end of FY), record actuals and estimate upcoming FY budget**
    1. Collect current FY monthly water revenue for all months
    2. Pull current FY actuals and budgeted estimates (debt service, acquisition credits, unencumbered funds from previous FY, budgeted gross revenues, fixed and variable operational expenses, non-sales revenues, transfers in as revenue from Rate Stabilization, R&R, and Other Funds)
    3. Pull previous FY actuals and re-calculate previous FY actual gross revenues (from total sales revenue, acquisition credits, non-sales revenue, net Rate Stabilization Fund transfer, end-of-FY deposit, and balance, R&R Fund balance, Utility Reserve Fund Balance)
    4. Check Fund conditions
       - If R&R Fund balance < 5% of previous FY gross revenue, a deposit is required in current FY to reach target
       - If Reserve Fund Balance < 10% of current FY gross revenues, a deposit is required to reach target
    5. Estimate CIP Fund deposit (as random number between 0.6-4% of current FY gross revenues plus a deeply uncertain multiplier)

Actuals equal to "observations" from Water Supply Modeling

Actuals assumed equal to Budgeted Amounts

Actuals differ from Budgeted Amounts by Deeply Uncertain Factor

Code Structure for 1 Realization – End of FY

- …
  - **If September (end of FY), record actuals and estimate upcoming FY budget**

6. Calculate Rate Covenant (calculate_RateCoverageRatio)
   - if Ratio < 1.25, record an annual "failure" and budget a needed deposit to the Reserve Fund Balance to meet target
7. Calculate Debt Covenant (calculate_DebtCoverageRatio)
   - If Ratio < 1.0, record annual "failure" and adjust needed transfer in as revenue from Rate Stabilization Fund to meet target
8. Check if current FY needed/estimate transfer in from Rate Stabilization Fund is under the "cap" (min. of 3% current gross revenues, unencumbered funds carried forward, previous FY deposit to Rate Stabilization Fund)
   - If budgeted transfer in exceeds cap, reduce transfer and increase transfers in from Other Funds to balance actual costs and revenues
9. Calculate final "true" gross revenues, net revenues, expenses before optional fund deposits, and budget surplus before optional fund deposits
10. Estimate fund deposits based on surplus
    - If surplus < 0 (deficit), no optional deposits to funds made, Reserve Fund Balance reduced to cover deficit
    - If surplus > 0, "needed" Reserve Fund deposits made, remaining surplus marked as unencumbered and/or deposited to Rate Stabilization Fund
11. Record actuals for ending FY, append to existing historical record

Code Structure for 1 Realization – End of FY

- …
  - **If September (end of FY), record actuals and estimate upcoming FY budget**
    12. **Estimate budget for upcoming FY about to start**
        1. If new infrastructure project was triggered, issue debt for it (add_NewDebt, assumed 30-year maturity, 4% interest rate)
        2. Estimate total debt service for upcoming FY (set_BudgetedDebtService, for now based on rough annual "cap" with preference to pay down principal on older issues first)
        3. If new project added, update budgeted costs to include new O&M costs
        4. Set acquisition credits owed
        5. Estimate fixed and variable operating expenses for next FY (based on fixed inflation rate of 3.3% annually)
        6. Get budgeted unencumbered carryover revenues (assumed to be 2.5% of current FY total sales revenue)
        7. Estimate TBC sales rate and revenue (assumed fixed for all future years at 2020 rate of $0.195/kgal)
        8. Estimate transfer in from Rate Stabilization Fund (initially, random number between $1.5M and 4% of current FY sales revenues, decreased if Rate Stabilization Fund balance falls below 8.5% of current FY gross revenue)
        9. Estimate R&R Fund transfer in and deposit (random numbers, but adjusted to ensure required R&R Fund balance)

Code Structure for 1 Realization – End of FY

- ↘ …
  - ↘ **If September (end of FY), record actuals and estimate upcoming FY budget**

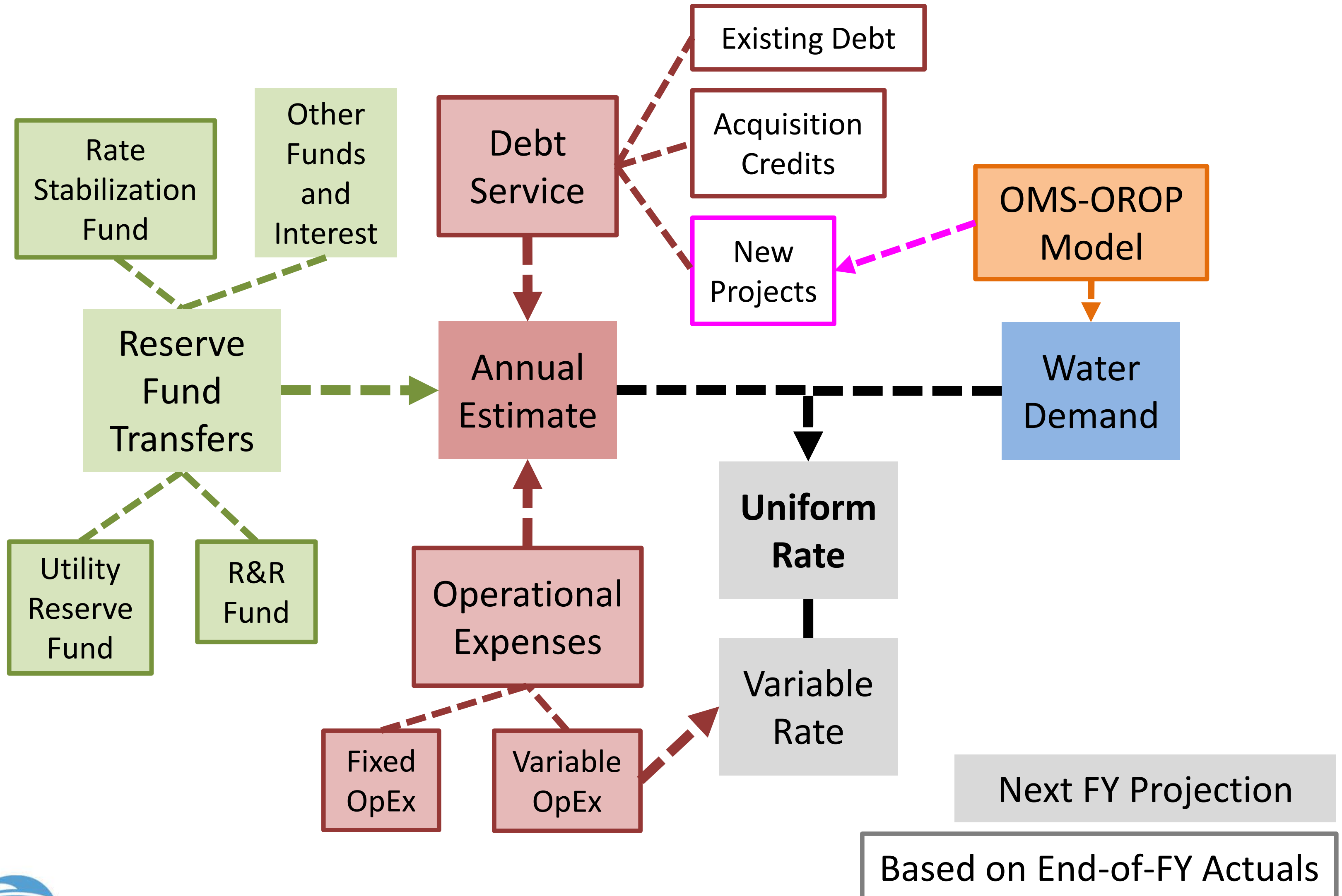
    12. Estimate budget for upcoming FY about to start

    10. Estimate income from interest
        (random, between $1.5-2M)
        Estimate budgeted deposits to Other Funds
        (random, between $200k-2M)
    11. Estimate Utility Reserve Fund deposits
        (only budgeted if previous year drew fund down significantly)
    12. Finalize Annual Estimate
    13. Estimate next FY water demand
        (using 1% annual growth rate from current FY deliveries)
    14. Estimate Uniform Rate (estimate_UniformRate)
        (potential to cap change in Rate between FYs by increasing Rate Stabilization transfers in)
    15. Estimate Variable Rate portion of Uniform Rate
        (Uniform Rate * Variable Cost fraction of Annual Estimate)
    16. Estimate budgeted sales revenues
    17. Calculate gross revenue, net revenue estimates for next FY
    18. Record next FY budgeted amounts, append to historical record of budgets

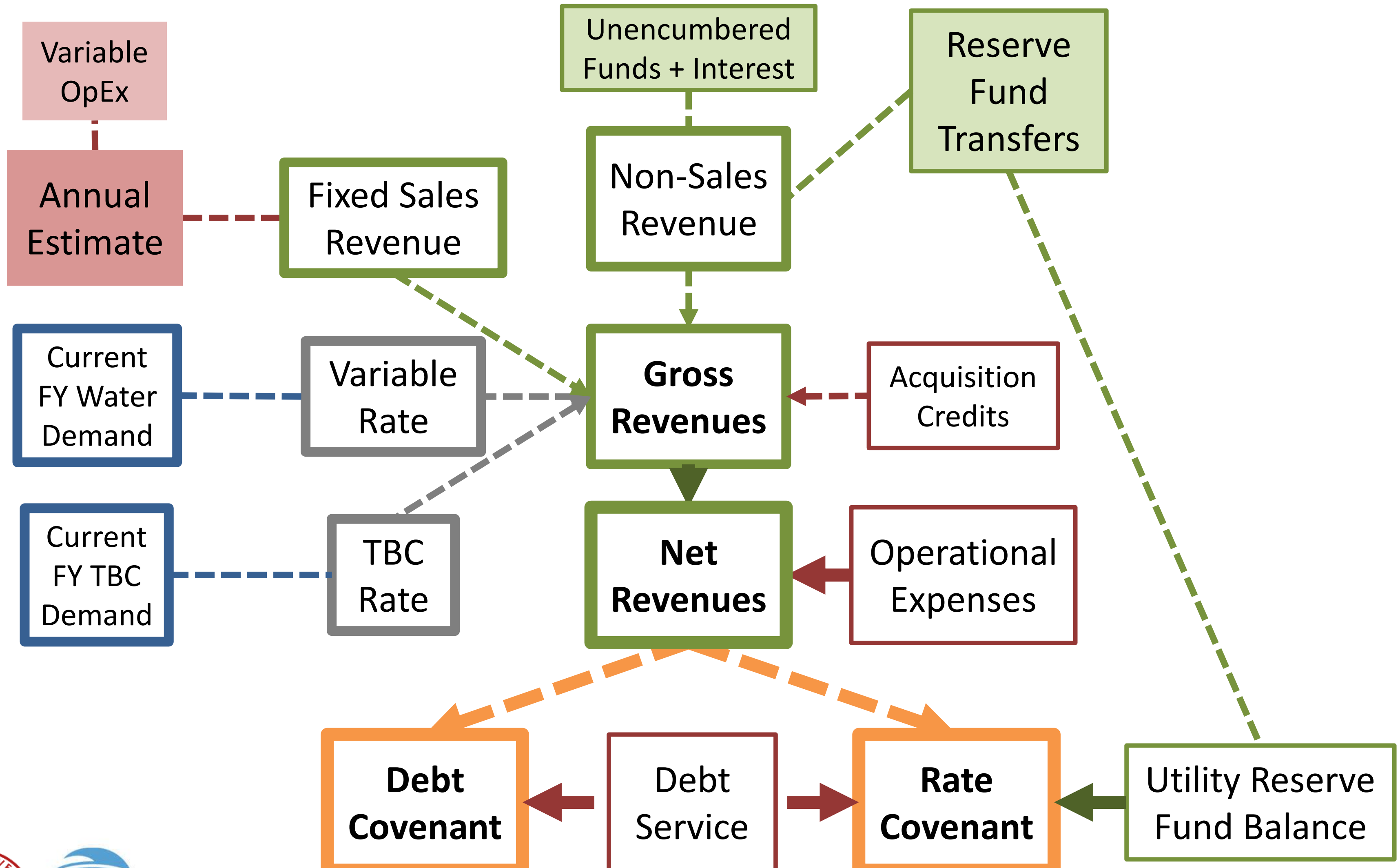
Code Structure for 1 Realization – Export

- **1 realization (2020-2040)**
  - **Enter realization function** (run_FinancialModelForSingleRealization)
    - **Enter annual loop from Jan 2020 – Dec 2039**
      - **Enter monthly loop Jan – Dec each calendar year**
        - **If September (end of FY), record actuals and estimate upcoming FY budget**
          - **…**
    - **Finish annual loop after Dec 2039, export results**

Uniform Rate Estimation



Calculating Covenants from Actuals



Model Details - Uncertainties / Assumptions

How financials are impacted by model design

Parameter or deeply uncertain factor

Random variability (within ranges)

Percentage of Budgeted Revenues carried forward as unencumbered funds

Randomness in fund transfers and interest income

Inflation (growth) rate of budgeted operational costs

Maturity, interest rate, repayment schedule of future and existing debt

Management influence on uniform rate setting and reserve fund transfers

Budgeted rate of water demand growth vs. “observed” record

Performance Metrics Review

A. Water supply
   1. LOS reliability: average realization fraction of days in failure
   2. LOS vulnerability: average realization 14+ day failure events
B. Environmental sustainability
   1. Average realization monitoring well level deficit (compared to permit level, aggregated by wellfield)
   2. Worst-case (99^{th} percentile) water table depletion (average monitoring deficit in worst year of realization)
C. Financial status
   1. Average net present cost of infrastructure expansion
   2. (proxy) Lower-bound "costs" of shortage mitigation

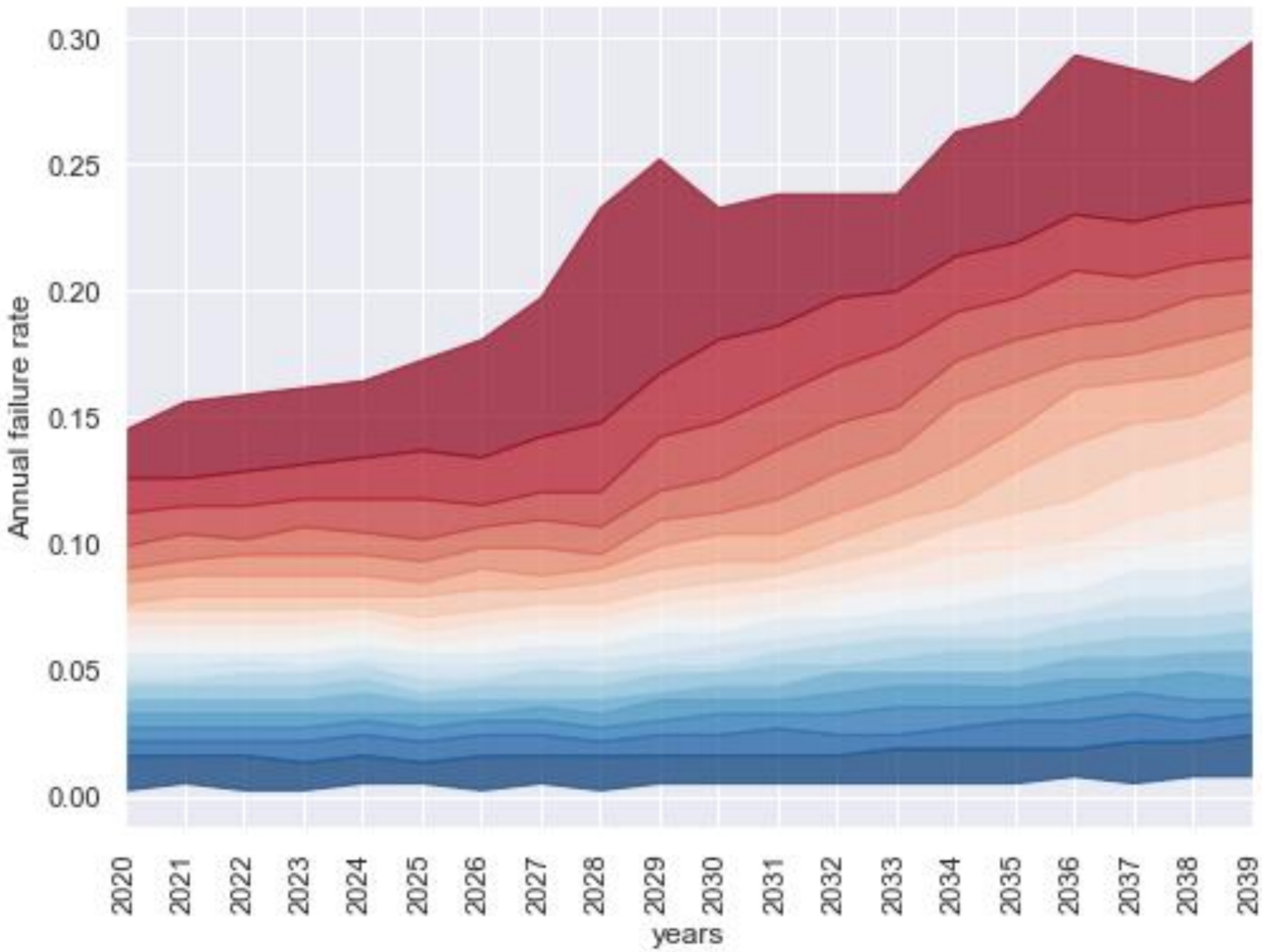
Performance Metrics Review

A. Water supply
   1. LOS reliability: average realization fraction of days in failure
   2. LOS vulnerability: average realization 14+ day failure events

B. Environmental sustainability
   1. Average realization monitoring well level deficit (compared to permit level, aggregated by wellfield)
   2. Worst-case (99th percentile) water table depletion (average monitoring deficit in worst year of realization)

C. Financial status
   1. Average net present cost of infrastructure expansion
   2. (proxy) Lower-bound “costs” of shortage mitigation

How is this plot generated?



Percentiles

Year	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%	...	90%	91%	92%	93%	94%	95%	96%	97%	98%	99%	100%
1	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.10	0.10	0.10	0.11	0.11	0.12	0.12	0.12	0.12	0.13	0.15
2	0.00	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02		0.13	0.13	0.13	0.14	0.15	0.15	0.16	0.16	0.17	0.29	0.32
3	0.00	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02		0.13	0.13	0.13	0.13	0.14	0.14	0.15	0.16	0.18	0.18	0.36
4	0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.16	0.16	0.16	0.17
5	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02		0.14	0.14	0.15	0.16	0.16	0.17	0.17	0.17	0.18	0.21	0.22
6	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03	0.03		0.12	0.13	0.14	0.14	0.14	0.15	0.18	0.20	0.26	0.30	0.31
7	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02		0.14	0.15	0.15	0.15	0.15	0.18	0.18	0.18	0.19	0.22	0.29
8	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02		0.13	0.14	0.14	0.14	0.16	0.18	0.19	0.32	0.32	0.36	0.40
9	0.00	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02		0.12	0.13	0.14	0.15	0.16	0.18	0.18	0.20	0.23	0.37	0.52
10	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	...	0.12	0.12	0.13	0.15	0.15	0.16	0.17	0.19	0.32	0.43	0.46
11	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.18	0.19	0.19	0.20	0.20	0.22	0.23	0.26	0.27	0.31	0.47
12	0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.19	0.19	0.19	0.20	0.23	0.24	0.25	0.27	0.27	0.30	0.37
13	0.00	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03		0.18	0.19	0.20	0.20	0.20	0.23	0.24	0.31	0.33	0.42	0.45
14	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02		0.21	0.21	0.23	0.25	0.25	0.26	0.27	0.33	0.34	0.35	0.44
15	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02		0.21	0.22	0.23	0.23	0.24	0.24	0.28	0.32	0.33	0.35	0.37
16	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03		0.21	0.22	0.22	0.22	0.23	0.25	0.27	0.28	0.37	0.38	0.47
17	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03		0.22	0.22	0.24	0.25	0.25	0.25	0.26	0.28	0.31	0.36	0.38
18	0.00	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.03		0.22	0.22	0.23	0.23	0.24	0.25	0.26	0.28	0.29	0.42	0.48
19	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03		0.22	0.22	0.23	0.23	0.24	0.25	0.25	0.27	0.34	0.46	0.51
20	0.00	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.03	0.03		0.21	0.21	0.21	0.22	0.22	0.23	0.25	0.26	0.31	0.34	0.59

Percentiles

Percentiles

Year	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%	...	90%	91%	92%	93%	94%	95%	96%	97%	98%	99%	100%
1	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.10	0.10	0.10	0.11	0.11	0.12	0.12	0.12	0.12	0.13	0.15
2	0.00	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02		0.13	0.13	0.13	0.14	0.15	0.15	0.16	0.16	0.17	0.29	0.32
3	0.00	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02		0.13	0.13	0.13	0.13	0.14	0.14	0.15	0.16	0.18	0.18	0.36
4	0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.16	0.16	0.16	0.17
5	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02		0.14	0.14	0.15	0.16	0.16	0.17	0.17	0.17	0.18	0.21	0.22
6	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03	0.03		0.12	0.13	0.14	0.14	0.14	0.15	0.18	0.20	0.26	0.30	0.31
7	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02		0.14	0.15	0.15	0.15	0.15	0.18	0.18	0.18	0.19	0.22	0.29
8	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02		0.13	0.14	0.14	0.14	0.16	0.18	0.19	0.32	0.32	0.36	0.40
9	0.00	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02		0.12	0.13	0.14	0.15	0.16	0.18	0.18	0.20	0.23	0.37	0.52
10	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	...	0.12	0.12	0.13	0.15	0.15	0.16	0.17	0.19	0.32	0.43	0.46
11	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.18	0.19	0.19	0.20	0.20	0.22	0.23	0.26	0.27	0.31	0.47
12	0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.19	0.19	0.19	0.20	0.23	0.24	0.25	0.27	0.27	0.30	0.37
13	0.00	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03		0.18	0.19	0.20	0.20	0.20	0.23	0.24	0.31	0.33	0.42	0.45
14	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02		0.21	0.21	0.23	0.25	0.25	0.26	0.27	0.33	0.34	0.35	0.44
15	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02		0.21	0.22	0.23	0.23	0.24	0.24	0.28	0.32	0.33	0.35	0.37
16	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03		0.21	0.22	0.22	0.22	0.23	0.25	0.27	0.28	0.37	0.38	0.47
17	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03		0.22	0.22	0.24	0.25	0.25	0.25	0.26	0.28	0.31	0.36	0.38
18	0.00	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.03		0.22	0.22	0.23	0.23	0.24	0.25	0.26	0.28	0.29	0.42	0.48
19	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03		0.22	0.22	0.23	0.23	0.24	0.25	0.25	0.27	0.34	0.46	0.51
20	0.00	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.03	0.03		0.21	0.21	0.21	0.22	0.22	0.23	0.25	0.26	0.31	0.34	0.59

Percentiles

Time

Year	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%	...	90%	91%	92%	93%	94%	95%	96%	97%	98%	99%	100%
1	0.01	0.01	0 01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.10	0.10	0.10	0.11	0.11	0.12	0.12	0.12	0.12	0.13	0.15
2	0.00	0.01	0 01	0.01	0.02	0.02	0.02	0.02	0.02	0.02		0.13	0.13	0.13	0.14	0.15	0.15	0.16	0.16	0.17	0.29	0.32
3	0.00	0.00	0 01	0.01	0.01	0.01	0.02	0.02	0.02	0.02		0.13	0.13	0.13	0.13	0.14	0.14	0.15	0.16	0.18	0.18	0.36
4	0.00	0.00	0 01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.16	0.16	0.16	0.17
5	0.01	0.01	0 01	0.01	0.02	0.02	0.02	0.02	0.02	0.02		0.14	0.14	0.15	0.16	0.16	0.17	0.17	0.17	0.18	0.21	0.22
6	0.01	0.01	0 01	0.02	0.02	0.02	0.02	0.03	0.03	0.03		0.12	0.13	0.14	0.14	0.14	0.15	0.18	0.20	0.26	0.30	0.31
7	0.01	0.01	0 01	0.01	0.01	0.02	0.02	0.02	0.02	0.02		0.14	0.15	0.15	0.15	0.15	0.18	0.18	0.18	0.19	0.22	0.29
8	0.01	0.01	0 01	0.01	0.01	0.01	0.02	0.02	0.02	0.02		0.13	0.14	0.14	0.14	0.16	0.18	0.19	0.32	0.32	0.36	0.40
9	0.00	0.01	0 01	0.02	0.02	0.02	0.02	0.02	0.02	0.02		0.12	0.13	0.14	0.15	0.16	0.18	0.18	0.20	0.23	0.37	0.52
10	0.00	0.01	0 01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	•••	0.12	0.12	0.13	0.15	0.15	0.16	0.17	0.19	0.32	0.43	0.46
11	0.00	0.01	0 01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.18	0.19	0.19	0.20	0.20	0.22	0.23	0.26	0.27	0.31	0.47
12	0.00	0.00	0 01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.19	0.19	0.19	0.20	0.23	0.24	0.25	0.27	0.27	0.30	0.37
13	0.00	0.00	0 01	0.01	0.01	0.01	0.02	0.02	0.02	0.03		0.18	0.19	0.20	0.20	0.20	0.23	0.24	0.31	0.33	0.42	0.45
14	0.01	0.02	0 02	0.02	0.02	0.02	0.02	0.02	0.02	0.02		0.21	0.21	0.23	0.25	0.25	0.26	0.27	0.33	0.34	0.35	0.44
15	0.01	0.01	0 01	0.02	0.02	0.02	0.02	0.02	0.02	0.02		0.21	0.22	0.23	0.23	0.24	0.24	0.28	0.32	0.33	0.35	0.37
16	0.01	0.01	0 01	0.01	0.02	0.02	0.02	0.02	0.03	0.03		0.21	0.22	0.22	0.22	0.23	0.25	0.27	0.28	0.37	0.38	0.47
17	0.01	0.01	0 01	0.01	0.01	0.02	0.02	0.02	0.02	0.03		0.22	0.22	0.24	0.25	0.25	0.25	0.26	0.28	0.31	0.36	0.38
18	0.00	0.01	0 01	0.01	0.02	0.02	0.02	0.03	0.03	0.03		0.22	0.22	0.23	0.23	0.24	0.25	0.26	0.28	0.29	0.42	0.48
19	0.01	0.01	0 01	0.01	0.02	0.02	0.02	0.02	0.03	0.03		0.22	0.22	0.23	0.23	0.24	0.25	0.25	0.27	0.34	0.46	0.51
20	0.00	0.01	0 01	0.02	0.02	0.02	0.03	0.03	0.03	0.03		0.21	0.21	0.21	0.22	0.22	0.23	0.25	0.26	0.31	0.34	0.59

Percentiles

Year	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%	...	90%	91%	92%	93%	94%	95%	96%	97%	98%	99%	100%
1	$\boxed{0.01}$	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.10	0.10	0.10	0.11	0.11	0.12	0.12	0.12	0.12	0.13	0.15
2	0.00	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02		0.13	0.13	0.13	0.14	0.15	0.15	0.16	0.16	0.17	0.29	0.32
3	0.00	0.00	0.01	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]		0.13	0.13	0.13	0.13	0.14	0.14	0.15	0.16	0.18	0.18	0.36
4	0.00	0.00	0.01	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]		0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.16	0.16	0.16	0.17
5	0.01	0.01	0.01	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]		0.14	0.14	0.15	0.16	0.16	0.17	0.17	0.17	0.18	0.21	0.22
6	0.01	0.01	0.01	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]		0.12	0.13	0.14	0.14	0.14	0.15	0.18	0.20	0.26	0.30	0.31
7	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02		0.14	0.15	0.15	0.15	0.15	0.18	0.18	0.18	0.19	0.22	0.29
8	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02		0.13	0.14	0.14	0.14	0.16	0.18	0.19	0.32	0.32	0.36	0.40
9	0.00	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02		0.12	0.13	0.14	0.15	0.16	0.18	0.18	0.20	0.23	0.37	0.52
10	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	...	0.12	0.12	0.13	0.15	0.15	0.16	0.17	0.19	0.32	0.43	0.46
11	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.18	0.19	0.19	0.20	0.20	0.22	0.23	0.26	0.27	0.31	0.47
12	0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.19	0.19	0.19	0.20	0.23	0.24	0.25	0.27	0.27	0.30	0.37
13	0.00	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03		0.18	0.19	0.20	0.20	0.20	0.23	0.24	0.31	0.33	0.42	0.45
14	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02		0.21	0.21	0.23	0.25	0.25	0.26	0.27	0.33	0.34	0.35	0.44
15	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02		0.21	0.22	0.23	0.23	0.24	0.24	0.28	0.32	0.33	0.35	0.37
16	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03		0.21	0.22	0.22	0.22	0.23	0.25	0.27	0.28	0.37	0.38	0.47
17	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03		0.22	0.22	0.24	0.25	0.25	0.25	0.26	0.28	0.31	0.36	0.38
18	0.00	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.03		0.22	0.22	0.23	0.23	0.24	0.25	0.26	0.28	0.29	0.42	0.48
19	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03		0.22	0.22	0.23	0.23	0.24	0.25	0.25	0.27	0.34	0.46	0.51
20	0.00	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.03	0.03		0.21	0.21	0.21	0.22	0.22	0.23	0.25	0.26	0.31	0.34	0.59

Lowest failure rate across 1000 realizations in **year 1**

Percentiles

Year	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
1	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02
2	0.00	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02
3	0.00	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02
4	0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02
5	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02
6	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03	0.03
7	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02
8	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02
9	0.00	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02
10	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02
11	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02
12	0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02
13	0.00	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03
14	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
15	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02
16	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03
17	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03
18	0.00	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.03
19	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03
20	0.00	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.03	0.03

...

Year	90%	91%	92%	93%	94%	95%	96%	97%	98%	99%	100%
1	0.10	0.10	0.10	0.11	0.11	0.12	0.12	0.12	0.12	0.13	$\boxed{0.15}$
2	0.13	0.13	0.13	0.14	0.15	0.15	0.16	0.16	0.17	[illegible]	0.32
3	0.13	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.18	0.18	0.36
4	0.13	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.16	0.16	0.17
5	0.14	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.18	0.21	0.22
6	0.12	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.26	0.30	0.31
7	0.14	0.15	0.15	0.15	0.15	0.18	0.18	0.18	0.19	0.22	0.29
8	0.13	0.14	0.14	0.14	0.16	0.18	0.19	0.32	0.32	0.36	0.40
9	0.12	0.13	0.14	0.15	0.16	0.18	0.18	0.20	0.23	0.37	0.52
10	0.12	0.12	0.13	0.15	0.15	0.16	0.17	0.19	0.32	0.43	0.46
11	0.18	0.19	0.19	0.20	0.20	0.22	0.23	0.26	0.27	0.31	0.47
12	0.19	0.19	0.19	0.20	0.23	0.24	0.25	0.27	0.27	0.30	0.37
13	0.18	0.19	0.20	0.20	0.20	0.23	0.24	0.31	0.33	0.42	0.45
14	0.21	0.21	0.23	0.25	0.25	0.26	0.27	0.33	0.34	0.35	0.44
15	0.21	0.22	0.23	0.23	0.24	0.24	0.28	0.32	0.33	0.35	0.37
16	0.21	0.22	0.22	0.22	0.23	0.25	0.27	0.28	0.37	0.38	0.47
17	0.22	0.22	0.24	0.25	0.25	0.25	0.26	0.28	0.31	0.36	0.38
18	0.22	0.22	0.23	0.23	0.24	0.25	0.26	0.28	0.29	0.42	0.48
19	0.22	0.22	0.23	0.23	0.24	0.25	0.25	0.27	0.34	0.46	0.51
20	0.21	0.21	0.21	0.22	0.22	0.23	0.25	0.26	0.31	0.34	0.59

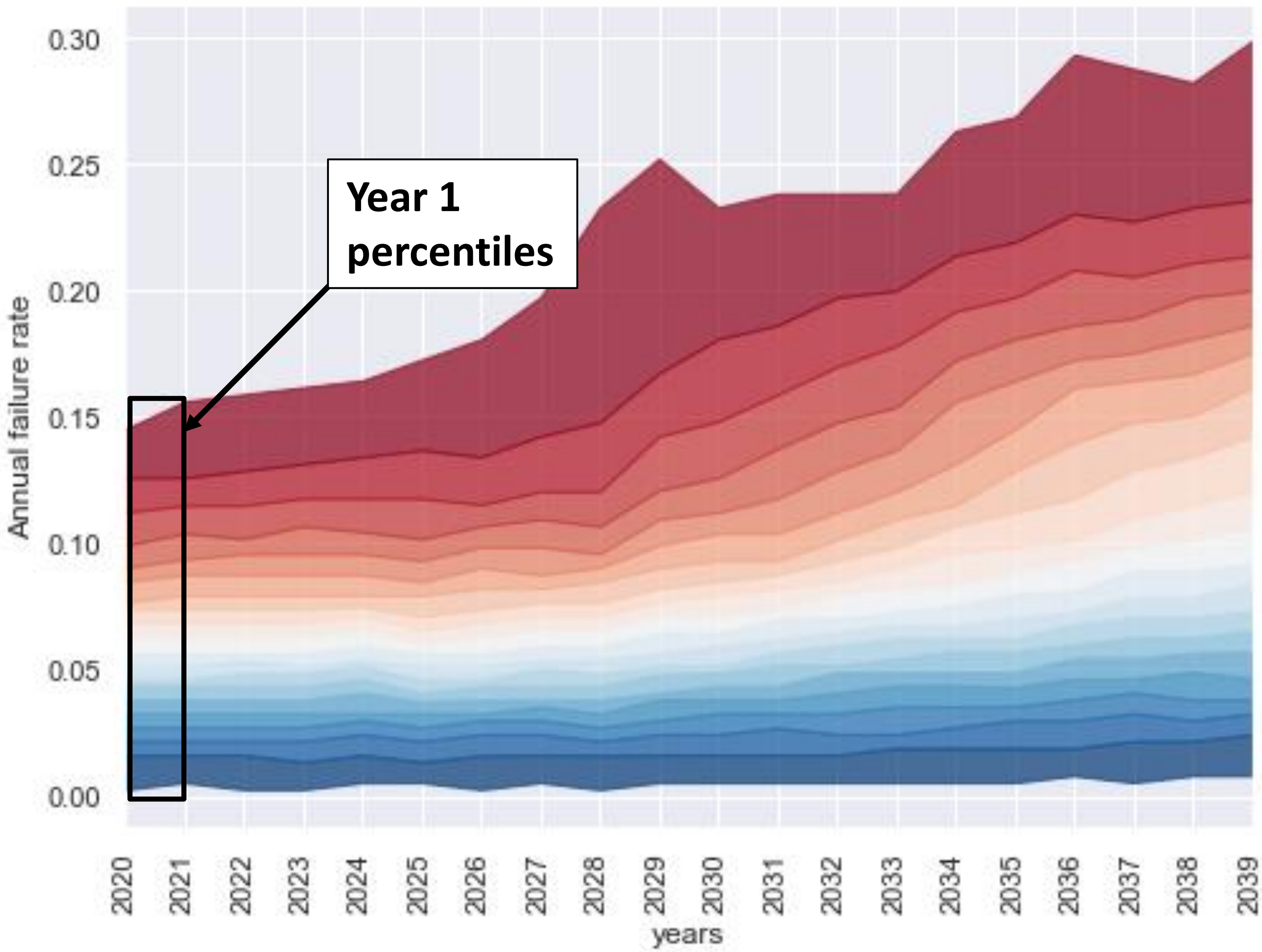
highest failure rate across 1000 realizations in **year 1**

Percentiles

Year	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%	…	90%	91%	92%	93%	94%	95%	96%	97%	98%	99%	100%
1	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.10	0.10	0.10	0.11	0.11	0.12	0.12	0.12	0.12	0.13	0.15
2	0.00	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02		0.13	0.13	0.13	0.14	0.15	0.15	0.16	0.16	0.17	0.29	0.32
3	0.00	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02		0.13	0.13	0.13	0.13	0.14	0.14	0.15	0.16	0.18	0.18	0.36
4	0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.16	0.16	0.16	0.17
5	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02				0.15	0.16	0.16	0.17	0.17	0.17	0.18	0.21	0.22
6	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03	0.03				0.14	0.14	0.14	0.15	0.18	0.20	0.26	0.30	0.31
7	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02				0.15	0.15	0.15	0.18	0.18	0.18	0.19	0.22	0.29
8	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02		0.13	0.14	0.14	0.14	0.16	0.18	0.19	0.32	0.32	0.36	0.40
9	0.00	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02		0.12	0.13	0.14	0.15	0.16	0.18	0.18	0.20	0.23	0.37	0.52
10	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	…	0.12	0.12	0.13	0.15	0.15	0.16	0.17	0.19	0.32	0.43	0.46
11	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.18	0.19	0.19	0.20	0.20	0.22	0.23	0.26	0.27	0.31	0.47
12	0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.19	0.19	0.19	0.20	0.23	0.24	0.25	0.27	0.27	0.30	0.37
13	0.00	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03		0.18	0.19	0.20	0.20	0.20	0.23	0.24	0.31	0.33	0.42	0.45
14	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02		0.21	0.21	0.23	0.25	0.25	0.26	0.27	0.33	0.34	0.35	0.44
15	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02		0.21	0.22	0.23	0.23	0.24	0.24	0.28	0.32	0.33	0.35	0.37
16	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03		0.21	0.22	0.22	0.22	0.23	0.25	0.27	0.28	0.37	0.38	0.47
17	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03		0.22	0.22	0.24	0.25	0.25	0.25	0.26	0.28	0.31	0.36	0.38
18	0.00	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.03		0.22	0.22	0.23	0.23	0.24	0.25	0.26	0.28	0.29	0.42	0.48
19	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03		0.22	0.22	0.23	0.23	0.24	0.25	0.25	0.27	0.34	0.46	0.51
20	0.00	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.03	0.03		0.21	0.21	0.21	0.22	0.22	0.23	0.25	0.26	0.31	0.34	0.59

Year 1 percentiles

How is this plot generated?

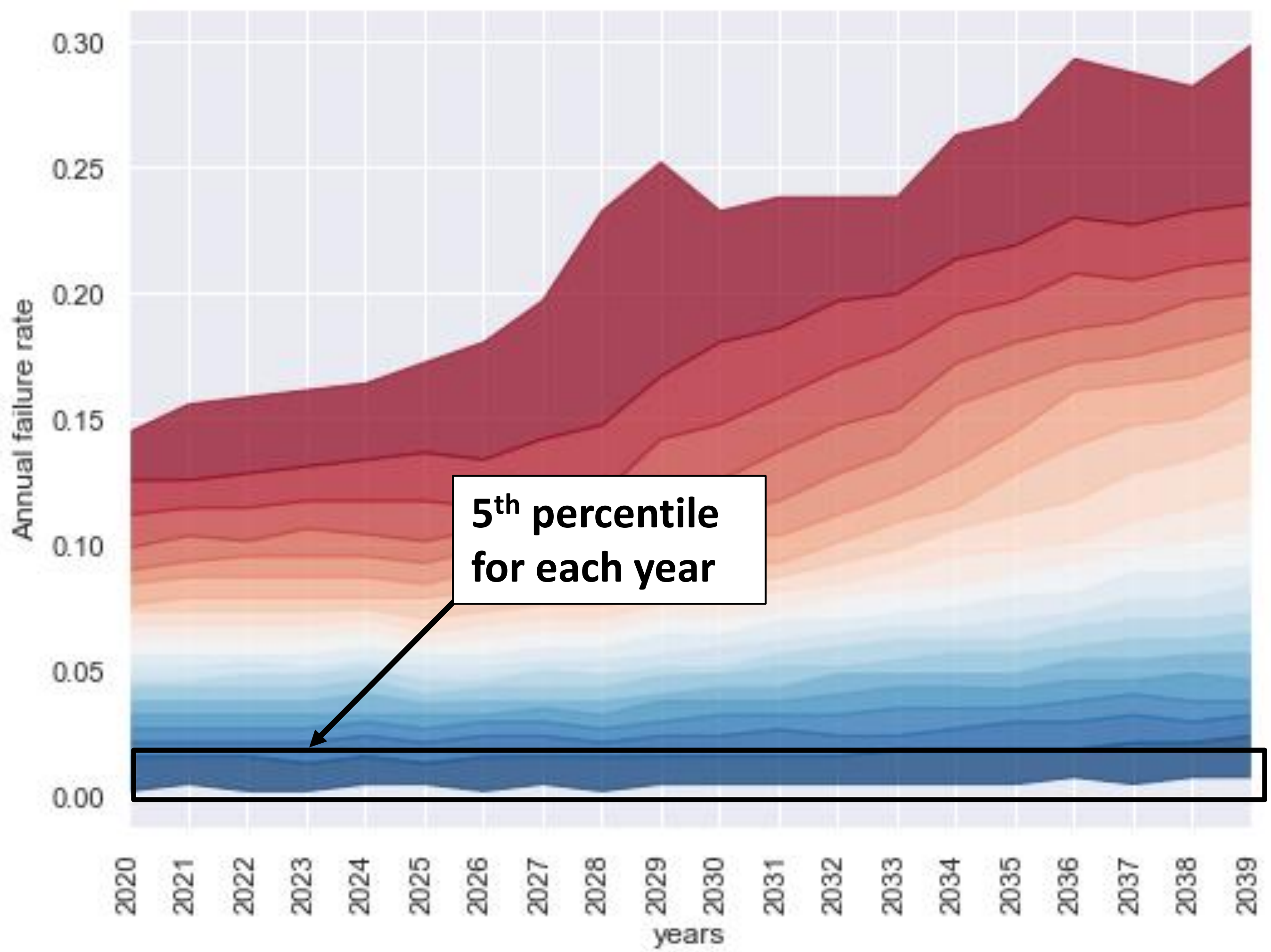


Percentiles

Year	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%	…	90%	91%	92%	93%	94%	95%	96%	97%	98%	99%	100%
1	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.10	0.10	0.10	0.11	0.11	0.12	0.12	0.12	0.12	0.13	0.15
2	0.00	0.01	0.01	0.01	0.02	0.02	[illegible]	[illegible]	[illegible]	[illegible]		0.13	0.13	0.13	0.14	0.15	0.15	0.16	0.16	0.17	0.29	0.32
3	0.00	0.00	0.01	0.01	0.01	0.01	[illegible]	[illegible]	[illegible]	[illegible]		0.13	0.13	0.13	0.13	0.14	0.14	0.15	0.16	0.18	0.18	0.36
4	0.00	0.00	0.01	0.01	0.01	0.01	[illegible]	[illegible]	[illegible]	[illegible]		0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.16	0.16	0.16	0.17
5	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02		0.14	0.14	0.15	0.16	0.16	0.17	0.17	0.17	0.18	0.21	0.22
6	0.01	0.01	0.01	0.02	0.02	[illegible]	0.02	0.03	0.03	0.03		0.12	0.13	0.14	0.14	0.14	0.15	0.18	0.20	0.26	0.30	0.31
7	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02		0.14	0.15	0.15	0.15	0.15	0.18	0.18	0.18	0.19	0.22	0.29
8	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02		0.13	0.14	0.14	0.14	0.16	0.18	0.19	0.32	0.32	0.36	0.40
9	0.00	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02		0.12	0.13	0.14	0.15	0.16	0.18	0.18	0.20	0.23	0.37	0.52
10	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	…	0.12	0.12	0.13	0.15	0.15	0.16	0.17	0.19	0.32	0.43	0.46
11	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.18	0.19	0.19	0.20	0.20	0.22	0.23	0.26	0.27	0.31	0.47
12	0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.19	0.19	0.19	0.20	0.23	0.24	0.25	0.27	0.27	0.30	0.37
13	0.00	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03		0.18	0.19	0.20	0.20	0.20	0.23	0.24	0.31	0.33	0.42	0.45
14	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02		0.21	0.21	0.23	0.25	0.25	0.26	0.27	0.33	0.34	0.35	0.44
15	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02		0.21	0.22	0.23	0.23	0.24	0.24	0.28	0.32	0.33	0.35	0.37
16	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03		0.21	0.22	0.22	0.22	0.23	0.25	0.27	0.28	0.37	0.38	0.47
17	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03		0.22	0.22	0.24	0.25	0.25	0.25	0.26	0.28	0.31	0.36	0.38
18	0.00	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.03		0.22	0.22	0.23	0.23	0.24	0.25	0.26	0.28	0.29	0.42	0.48
19	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03		0.22	0.22	0.23	0.23	0.24	0.25	0.25	0.27	0.34	0.46	0.51
20	0.00	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.03	0.03		0.21	0.21	0.21	0.22	0.22	0.23	0.25	0.26	0.31	0.34	0.59

5^{th} percentile for each year

How is this plot generated?



Percentiles

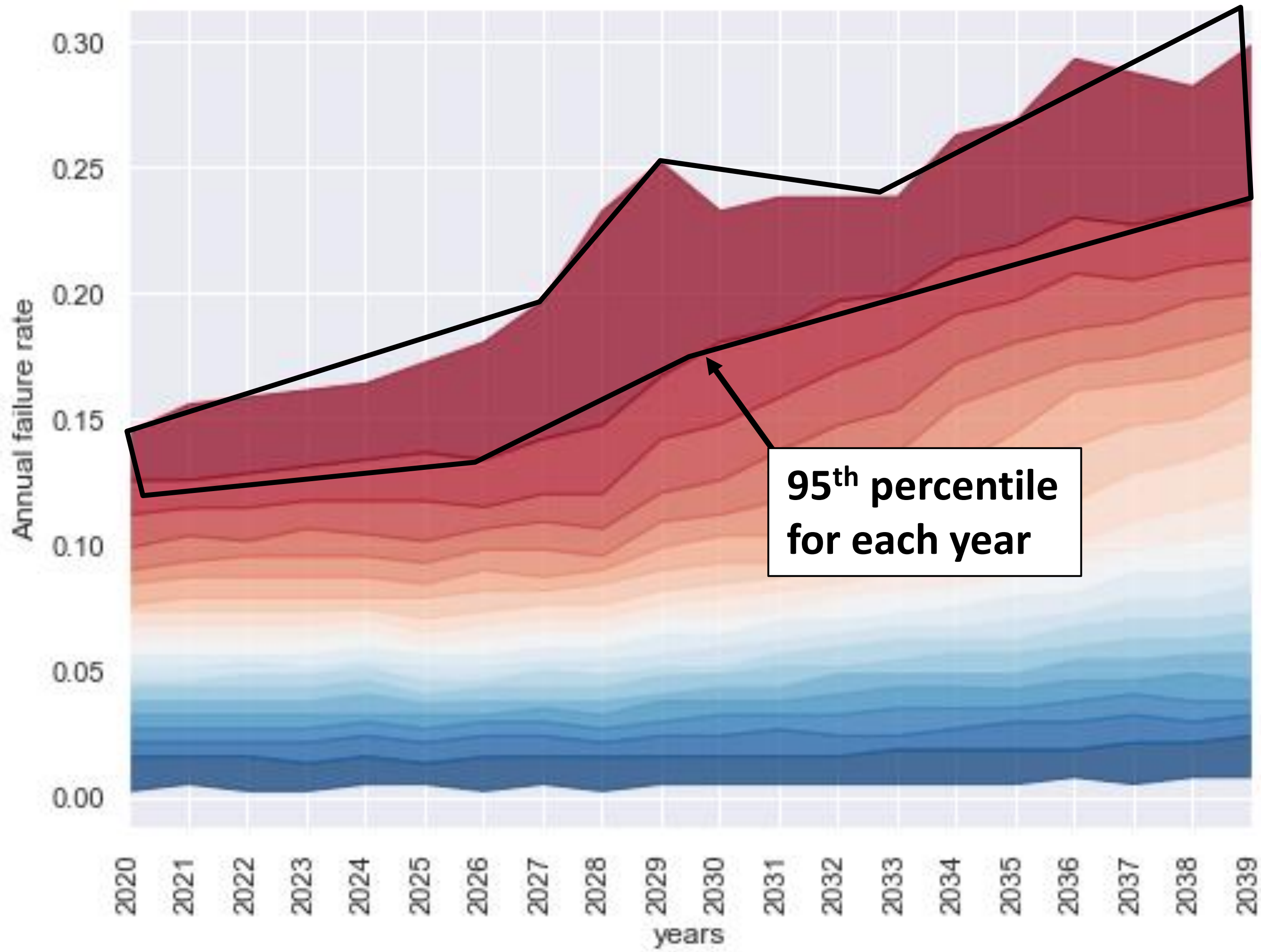
Year	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
1	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02
2	0.00	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02
3	0.00	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02
4	0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02
5	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02
6	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03	0.03
7	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02
8	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02
9	0.00	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02
10	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02
11	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02
12	0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02
13	0.00	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03
14	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
15	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02
16	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03
17	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03
18	0.00	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.03
19	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03
20	0.00	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.03	0.03

…

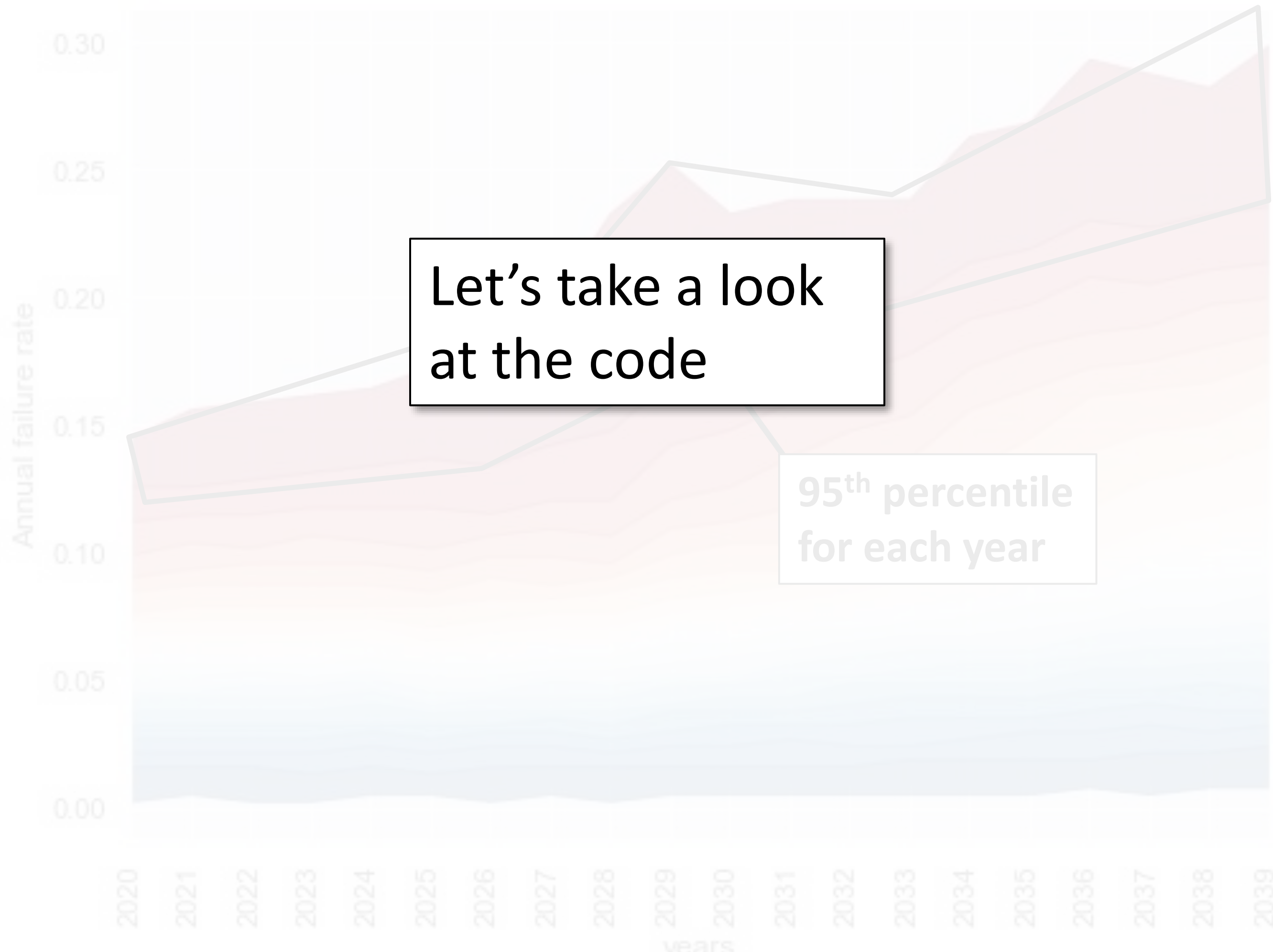
Year	90%	91%	92%	93%	94%	95%	96%	97%	98%	99%	100%
1	0.10	0.10	0.10	0.11	0.11	0.12	0.12	0.12	0.12	0.13	0.15
2	[illegible]	[illegible]	[illegible]	[illegible]	0.15	0.15	0.16	0.16	0.17	0.29	0.32
3	[illegible]	[illegible]	[illegible]	[illegible]	0.14	0.14	0.15	0.16	0.18	0.18	0.36
4	[illegible]	[illegible]	[illegible]	[illegible]	0.14	0.14	0.14	0.16	0.16	0.16	0.17
5	[illegible]	[illegible]	[illegible]	[illegible]	0.16	0.17	0.17	0.17	0.18	0.21	0.22
6	0.12	0.13	0.14	0.14	0.14	0.15	0.18	0.20	0.26	0.30	0.31
7	0.14	0.15	0.15	0.15	0.15	0.18	0.18	0.18	0.19	0.22	0.29
8	0.13	0.14	0.14	0.14	0.16	0.18	0.19	0.32	0.32	0.36	0.40
9	0.12	0.13	0.14	0.15	0.16	0.18	0.18	0.20	0.23	0.37	0.52
10	0.12	0.12	0.13	0.15	0.15	0.16	0.17	0.19	0.32	0.43	0.46
11	0.18	0.19	0.19	0.20	0.20	0.22	0.23	0.26	0.27	0.31	0.47
12	0.19	0.19	0.19	0.20	0.23	0.24	0.25	0.27	0.27	0.30	0.37
13	0.18	0.19	0.20	0.20	0.20	0.23	0.24	0.31	0.33	0.42	0.45
14	0.21	0.21	0.23	0.25	0.25	0.26	0.27	0.33	0.34	0.35	0.44
15	0.21	0.22	0.23	0.23	0.24	0.24	0.28	0.32	0.33	0.35	0.37
16	0.21	0.22	0.22	0.22	0.23	0.25	0.27	0.28	0.37	0.38	0.47
17	0.22	0.22	0.24	0.25	0.25	0.25	0.26	0.28	0.31	0.36	0.38
18	0.22	0.22	0.23	0.23	0.24	0.25	0.26	0.28	0.29	0.42	0.48
19	0.22	0.22	0.23	0.23	0.24	0.25	0.25	0.27	0.34	0.46	0.51
20	0.21	0.21	0.21	0.22	0.22	0.23	0.25	0.26	0.31	0.34	0.59

95^{th} percentile for each year

How is this plot generated?



How is this plot generated?



Let's take a look at the code

time_varying_objectives.py

```python
def plotLOSQuantiles(n_rel, figureName):

    # load each cleaned AMPL outfile and store in a list
    all_ampl = list()

    for rel in range(1,n_rel):
        if rel < 10:
            num_str = '000'+ str(rel)
        elif rel < 100:
            num_str = '00' + str(rel)
        else:
            num_str = '0' + str(rel)
        realization_data = pd.read_csv('Cleaned_zip/cleaned/ampl_' + num_str +
                                       '.csv')#, usecols=AMPL_cols)
        realization_data.fillna(0, inplace=True)

        all_ampl.append(realization_data)

    # Calculate LOS metrics for each year (missing leap years, fix later?)
    annual_LOS_rel = np.zeros([n_rel, 20])
    annual_LOS_vul = np.zeros([n_rel, 20])

    # loop through each realization
    for rel in range(0, n_rel-1):
        current_rel = all_ampl[rel]

        for day in np.linspace(0, 6935, 20):
            day = int(day)
            year = int(day/365)
            current_year = current_rel.iloc[day:day+365]
            annual_LOS_rel[rel, year], annual_LOS_vul[rel, year] = calculateLevelOfService(current_year)


    # Calculate percentiles across realizations
    LOS_rel = np.zeros([20,100])
    LOS_vul = np.zeros([20,100])
    # Extract 90th percentile from each LOS
    for year in range(0,20):
        for p in range(0,100):
            LOS_rel[year,p] = np.percentile(annual_LOS_rel[:,year], (p+1))
            LOS_vul[year,p] = np.percentile(annual_LOS_vul[:,year], (p+1))
```

```python
    # plot time varying distribution
    cmap = matplotlib.cm.get_cmap("RdBu_r")
    fig = plt.figure(figsize=(8,6))
    ax = fig.add_subplot(1,1,1)
    for i in np.linspace(5,95,19):
        i = int(i)
        ax.fill_between(range(2020,2040), LOS_rel[:,i-5], LOS_rel[:,i],
                        color=cm.RdBu_r((i-1)/100.0), alpha=0.75, edgecolor='none')
    ax.set_xticks(range(2020,2040))
    ax.set_xticklabels(range(2020,2040), rotation='vertical')
    ax.set_xlim([2020,2039])
    plt.xlabel('years')
    plt.ylabel('Annual failure rate')

    plt.savefig(figureName, bbox_inches= 'tight')

    return
```

time_varying_objectives.py

Input: number of realizations, desired name of figure

```python
def plotLOSQuantiles(n_rel, figureName):

    # Load each cleaned AMPL outfile and store in a list
    all_ampl = list()

    for rel in range(1,n_rel):
        if rel < 10:
            num_str = '000'+ str(rel)
        elif rel < 100:
            num_str = '00' + str(rel)
        else:
            num_str = '0' + str(rel)
        realization_data = pd.read_csv('Cleaned_zip/cleaned/ampl_' + num_str +
                                      '.csv')#, usecols=AMPL_cols)
        realization_data.fillna(0, inplace=True)

        all_ampl.append(realization_data)

    # Calculate LOS metrics for each year (missing leap years, fix later?)
    annual_LOS_rel = np.zeros([n_rel, 20])
    annual_LOS_vul = np.zeros([n_rel, 20])

    # loop through each realization
    for rel in range(0, n_rel-1):
        current_rel = all_ampl[rel]

        for day in np.linspace(0, 6935, 20):
            day = int(day)
            year = int(day/365)
            current_year = current_rel.iloc[day:day+365]
            annual_LOS_rel[rel, year], annual_LOS_vul[rel, year] = calculateLevelOfService(current_year)


    # Calculate percentiles across realizations
    LOS_rel = np.zeros([20,100])
    LOS_vul = np.zeros([20,100])
    # Extract 90th percentile from each LOS
    for year in range(0,20):
        for p in range(0,100):
            LOS_rel[year,p] = np.percentile(annual_LOS_rel[:,year], (p+1))
            LOS_vul[year,p] = np.percentile(annual_LOS_vul[:,year], (p+1))
```

```python
    # plot time varying distribution
    cmap = matplotlib.cm.get_cmap("RdBu_r")
    fig = plt.figure(figsize=(8,6))
    ax = fig.add_subplot(1,1,1)
    for i in np.linspace(5,95,19):
        i = int(i)
        ax.fill_between(range(2020,2040), LOS_rel[:,i-5], LOS_rel[:,i],
                        color=cm.RdBu_r((i-1)/100.0), alpha=0.75, edgecolor='none')
    ax.set_xticks(range(2020,2040))
    ax.set_xticklabels(range(2020,2040), rotation='vertical')
    ax.set_xlim([2020,2039])
    plt.xlabel('years')
    plt.ylabel('Annual failure rate')

    plt.savefig(figureName, bbox_inches= 'tight')

    return
```

time_varying_objectives.py

Step 1: Read AMPL output

```python
def plotLOSQuantiles(n_rel, figureName):

    # load each cleaned AMPL outfile and store in a list
    all_ampl = list()

    for rel in range(1,n_rel):
        if rel < 10:
            num_str = '000'+ str(rel)
        elif rel < 100:
            num_str = '00' + str(rel)
        else:
            num_str = '0' + str(rel)
        realization_data = pd.read_csv('Cleaned_zip/cleaned/ampl_' + num_str +
                                       '.csv')#, usecols=AMPL_cols)
        realization_data.fillna(0, inplace=True)

        all_ampl.append(realization_data)

    # Calculate LOS metrics for each year (missing leap years, fix later?)
    annual_LOS_rel = np.zeros([n_rel, 20])
    annual_LOS_vul = np.zeros([n_rel, 20])

    # loop through each realization
    for rel in range(0, n_rel-1):
        current_rel = all_ampl[rel]

        for day in np.linspace(0, 6935, 20):
            day = int(day)
            year = int(day/365)
            current_year = current_rel.iloc[day:day+365]
            annual_LOS_rel[rel, year], annual_LOS_vul[rel, year] = calculateLevelOfService(current_year)


    # Calculate percentiles across realizations
    LOS_rel = np.zeros([20,100])
    LOS_vul = np.zeros([20,100])
    # Extract 90th percentile from each LOS
    for year in range(0,20):
        for p in range(0,100):
            LOS_rel[year,p] = np.percentile(annual_LOS_rel[:,year], (p+1))
            LOS_vul[year,p] = np.percentile(annual_LOS_vul[:,year], (p+1))
```

```python
    # plot time varying distribution
    .cm.get_cmap("RdBu_r")
    figsize=(8,6))
    lot(1,1,1)
    ace(5,95,19):
        i = int(i)
        ax.fill_between(range(2020,2040), LOS_rel[:,i-5], LOS_rel[:,i],
                        color=cm.RdBu_r((i-1)/100.0), alpha=0.75, edgecolor='none')
    ax.set_xticks(range(2020,2040))
    ax.set_xticklabels(range(2020,2040), rotation='vertical')
    ax.set_xlim([2020,2039])
    plt.xlabel('years')
    plt.ylabel('Annual failure rate')

    plt.savefig(figureName, bbox_inches= 'tight')

    return
```

time_varying_objectives.py

```python
def plotLOSQuantiles(n_rel, figureName):

    # Load each cleaned AMPL outfile and store in a list
    all_ampl = list()

    for rel in range(1,n_rel):
        if rel < 10:
            num_str = '000'+ str(rel)
        elif rel < 100:
            num_str = '00' + str(rel)
        else:
            num_str = '0' + str(rel)
        realization_data = pd.read_csv('Cleaned_zip/cleaned/ampl_' + num_str +
                                       '.csv')#, usecols=AMPL_cols)
        realization_data.fillna(0, inplace=True)

        all_ampl.append(realization_data)

    # Calculate LOS metrics for each year (missing leap years, fix
    annual_LOS_rel = np.zeros([n_rel, 20])
    annual_LOS_vul = np.zeros([n_rel, 20])

    # Loop through each realization
    for rel in range(0, n_rel-1):
        current_rel = all_ampl[rel]

        for day in np.linspace(0, 6935, 20):
            day = int(day)
            year = int(day/365)
            current_year = current_rel.iloc[day:day+365]
            annual_LOS_rel[rel, year], annual_LOS_vul[rel, year] = calculateLevelOfService(current_year)


    # Calculate percentiles across realizations
    LOS_rel = np.zeros([20,100])
    LOS_vul = np.zeros([20,100])
    # Extract 90th percentile from each LOS
    for year in range(0,20):
        for p in range(0,100):
            LOS_rel[year,p] = np.percentile(annual_LOS_rel[:,year], (p+1))
            LOS_vul[year,p] = np.percentile(annual_LOS_vul[:,year], (p+1))
```

```python
    # plot time varying distribution
    cmap = matplotlib.cm.get_cmap("RdBu_r")
    fig = plt.figure(figsize=(8,6))
    ax = fig.add_subplot(1,1,1)
    for i in np.linspace(5,95,19):
        i = int(i)
        ax.fill_between(range(2020,2040), LOS_rel[:,i-5], LOS_rel[:,i],
                        color=cm.RdBu_r((i-1)/100.0), alpha=0.75, edgecolor='none')
    ax.set_xticks(range(2020,2040))
    ax.set_xticklabels(range(2020,2040), rotation='vertical')
    ax.set_xlim([2020,2039])
    plt.xlabel('years')
    plt.ylabel('Annual failure rate')

    plt.savefig(figureName, bbox_inches= 'tight')

    return
```

Step 2: Loop through each realization,

time_varying_objectives.py

```python
def plotLOSQuantiles(n_rel, figureName):

    # Load each cleaned AMPL outfile and store in a list
    all_ampl = list()

    for rel in range(1,n_rel):
        if rel < 10:
            num_str = '000'+ str(rel)
        elif rel < 100:
            num_str = '00' + str(rel)
        else:
            num_str = '0' + str(rel)
        realization_data = pd.read_csv('Cleaned_zip/cleaned/ampl_' + num_str +
                                       '.csv')#, usecols=AMPL_cols)
        realization_data.fillna(0, inplace=True)

        all_ampl.append(realization_data)

    # Calculate LOS metrics for each year (missing leap years, fix
    annual_LOS_rel = np.zeros([n_rel, 20])
    annual_LOS_vul = np.zeros([n_rel, 20])

    # Loop through each realization
    for rel in range(0, n_rel-1):
        current_rel = all_ampl[rel]

        for day in np.linspace(0, 6935, 20):
            day = int(day)
            year = int(day/365)
            current_year = current_rel.iloc[day:day+365]
            annual_LOS_rel[rel, year], annual_LOS_vul[rel, year] = calculateLevelOfService(current_year)

    # Calculate percentiles across realizations
    LOS_rel = np.zeros([20,100])
    LOS_vul = np.zeros([20,100])
    # Extract 90th percentile from each LOS
    for year in range(0,20):
        for p in range(0,100):
            LOS_rel[year,p] = np.percentile(annual_LOS_rel[:,year], (p+1))
            LOS_vul[year,p] = np.percentile(annual_LOS_vul[:,year], (p+1))
```

```python
    # plot time varying distribution
    cmap = matplotlib.cm.get_cmap("RdBu_r")
    fig = plt.figure(figsize=(8,6))
    ax = fig.add_subplot(1,1,1)
    for i in np.linspace(5,95,19):
        i = int(i)
        ax.fill_between(range(2020,2040), LOS_rel[:,i-5], LOS_rel[:,i],
                        color=cm.RdBu_r((i-1)/100.0), alpha=0.75, edgecolor='none')
    ax.set_xticks(range(2020,2040))
    ax.set_xticklabels(range(2020,2040), rotation='vertical')
    ax.set_xlim([2020,2039])
    plt.xlabel('years')
    plt.ylabel('Annual failure rate')

    plt.savefig(figureName, bbox_inches= 'tight')

    return
```

Step 2: Loop through each realization, and each year

time_varying_objectives.py

Step 2: Loop through each realization, and each year, and calculate LOS

```python
def plotLOSQuantiles(n_rel, figureName):

    # load each cleaned AMPL outfile and store in a list
    all_ampl = list()

    for rel in range(1,n_rel):
        if rel < 10:
            num_str = '000'+ str(rel)
        elif rel < 100:
            num_str = '00' + str(rel)
        else:
            num_str = '0' + str(rel)
        realization_data = pd.read_csv('Cleaned_zip/cleaned/ampl_' + num_str +
                                       '.csv')#, usecols=AMPL_cols)
        realization_data.fillna(0, inplace=True)

        all_ampl.append(realization_data)

    # Calculate LOS metrics for each year (missing leap years, fix
    annual_LOS_rel = np.zeros([n_rel, 20])
    annual_LOS_vul = np.zeros([n_rel, 20])

    # loop through each realization
    for rel in range(0, n_rel-1):
        current_rel = all_ampl[rel]

        for day in np.linspace(0, 6935, 20):
            day = int(day)
            year = int(day/365)
            current_year = current_rel.iloc[day:day+365]
            annual_LOS_rel[rel, year], annual_LOS_vul[rel, year] = calculateLevelOfService(current_year)

    # Calculate percentiles across realizations
    LOS_rel = np.zeros([20,100])
    LOS_vul = np.zeros([20,100])
    # Extract 90th percentile from each LOS
    for year in range(0,20):
        for p in range(0,100):
            LOS_rel[year,p] = np.percentile(annual_LOS_rel[:,year], (p+1))
            LOS_vul[year,p] = np.percentile(annual_LOS_vul[:,year], (p+1))
```

```python
    # plot time varying distribution
    cmap = matplotlib.cm.get_cmap("RdBu_r")
    fig = plt.figure(figsize=(8,6))
    ax = fig.add_subplot(1,1,1)
    for i in np.linspace(5,95,19):
        i = int(i)
        ax.fill_between(range(2020,2040), LOS_rel[:,i-5], LOS_rel[:,i],
                        color=cm.RdBu_r((i-1)/100.0), alpha=0.75, edgecolor='none')
    ax.set_xticks(range(2020,2040))
    ax.set_xticklabels(range(2020,2040), rotation='vertical')
    ax.set_xlim([2020,2039])
    plt.xlabel('years')
    plt.ylabel('Annual failure rate')

    plt.savefig(figureName, bbox_inches= 'tight')

    return
```

calculateLevelOfService.py

```python
def calculateLevelOfService(AMPL_cleaned_data):
    # call "get" functions to pull data necessary to calculate objective
    # statistics for the realization
    daily_CWUP_overage_vector, \
    daily_SCH_overage_vector, \
    daily_BUD_overage_vector = getGWPermitViolations(AMPL_cleaned_data)

    daily_Alafia_slack_vector, \
    daily_Reservoir_slack_vector, \
    daily_TBC_slack_vector = getSWPermitViolations(AMPL_cleaned_data)

    # calculate reliability as the fraction of days without permit violations
    # and take count of longest stretch of consecutive days of violation
    # (if greater than 365, entire realization is a failure, which is a good
    # way to track firm yield on the supply under stationary demand realizations
    # but may not function well under transient conditions to track failure)
    violation_tracker = np.stack((daily_CWUP_overage_vector,
                                  daily_SCH_overage_vector,
                                  daily_BUD_overage_vector,
                                  daily_Alafia_slack_vector,
                                  daily_Reservoir_slack_vector,
                                  daily_TBC_slack_vector), axis = 1)
```

```python
    # total daily violations for the region
    daily_violation_sum = np.nansum(violation_tracker, axis = 1)

    # calculate basic ratio of days in violation to total days
    realization_level_of_service_reliability = sum(
            daily_violation_sum > 0) / len(daily_violation_sum)

    # length of all violation events
    import itertools
    length_of_events = [sum(1 for _ in group) \
                        for key, group in \
                        itertools.groupby(daily_violation_sum > 0) if key]

    # check if largest event is at least 365 days
    if len(length_of_events) > 0:
        if max(length_of_events) > 364:
            print('Realization in failure')

    # calculate number of failure events (14+ consec days of failure)
    realization_count_of_failure_events_vulnerability = len(
            [True for x in length_of_events if x > 13])

    return [realization_level_of_service_reliability,
            realization_count_of_failure_events_vulnerability]
```

calculateLevelOfService.py

Input: Formatted AMPL data

```python
def calculateLevelOfService(AMPL_cleaned_data):
    # call "get" functions to pull data necessary to calculate objective
    # statistics for the realization
    daily_CWUP_overage_vector, \
    daily_SCH_overage_vector, \
    daily_BUD_overage_vector = getGWPermitViolations(AMPL_cleaned_data)

    daily_Alafia_slack_vector, \
    daily_Reservoir_slack_vector, \
    daily_TBC_slack_vector = getSWPermitViolations(AMPL_cleaned_data)

    # calculate reliability as the fraction of days without permit violations
    # and take count of longest stretch of consecutive days of violation
    # (if greater than 365, entire realization is a failure, which is a good
    # way to track firm yield on the supply under stationary demand realizations
    # but may not function well under transient conditions to track failure)
    violation_tracker = np.stack((daily_CWUP_overage_vector,
                                  daily_SCH_overage_vector,
                                  daily_BUD_overage_vector,
                                  daily_Alafia_slack_vector,
                                  daily_Reservoir_slack_vector,
                                  daily_TBC_slack_vector), axis = 1)

    # total daily violations for the region
    daily_violation_sum = np.nansum(violation_tracker, axis = 1)

    # calculate basic ratio of days in violation to total days
    realization_level_of_service_reliability = sum(
            daily_violation_sum > 0) / len(daily_violation_sum)

    # length of all violation events
    import itertools
    length_of_events = [sum(1 for _ in group) \
                        for key, group in \
                        itertools.groupby(daily_violation_sum > 0) if key]

    # check if largest event is at least 365 days
    if len(length_of_events) > 0:
        if max(length_of_events) > 364:
            print('Realization in failure')

    # calculate number of failure events (14+ consec days of failure)
    realization_count_of_failure_events_vulnerability = len(
            [True for x in length_of_events if x > 13])

    return [realization_level_of_service_reliability,
            realization_count_of_failure_events_vulnerability]
```

calculateLevelOfService.py

```python
def calculateLevelOfService(AMPL_cleaned_data):
    # call "get" functions to pull data necessary to calculate objective
    # statistics for the realization
    daily_CWUP_overage_vector, \
    daily_SCH_overage_vector, \
    daily_BUD_overage_vector = getGWPermitViolations(AMPL_cleaned_data)

    daily_Alafia_slack_vector, \
    daily_Reservoir_slack_vector, \
    daily_TBC_slack_vector = getSWPermitViolations(AMPL_cleaned_data)

    # calculate reliability as the fraction of days without permit violations
    # and take count of longest stretch of consecutive days of violation
    # (if greater than 365, entire realization is a failure, which is a good
    # way to track firm yield on the supply under stationary demand realizations
    # but may not function well under transient conditions to track failure)
    violation_tracker = np.stack((daily_CWUP_overage_vector,
                                  daily_SCH_overage_vector,
                                  daily_BUD_overage_vector,
                                  daily_Alafia_slack_vector,
                                  daily_Reservoir_slack_vector,
                                  daily_TBC_slack_vector), axis = 1)

    # total daily violations for the region
    daily_violation_sum = np.nansum(violation_tracker, axis = 1)

    # calculate basic ratio of days in violation to total days
    realization_level_of_service_reliability = sum(
            daily_violation_sum > 0) / len(daily_violation_sum)

    # length of all violation events
    import itertools
    length_of_events = [sum(1 for _ in group) \
                        for key, group in \
                        itertools.groupby(daily_violation_sum > 0) if key]

    # check if largest event is at least 365 days
    if len(length_of_events) > 0:
        if max(length_of_events) > 364:
            print('Realization in failure')

    # calculate number of failure events (14+ consec days of failure)
    realization_count_of_failure_events_vulnerability = len(
            [True for x in length_of_events if x > 13])

    return [realization_level_of_service_reliability,
            realization_count_of_failure_events_vulnerability]
```

Output: LOS reliability and vulnerability

calculateLevelOfService.py

Step 1: Calculate groundwater violations (CWUP, SCH, BUD)

```python
def calculateLevelOfService(AMPL_cleaned_data):
    # call "get" functions to pull data necessary to calculate objective
    # statistics for the realization
    daily_CWUP_overage_vector, \
    daily_SCH_overage_vector, \
    daily_BUD_overage_vector = getGWPermitViolations(AMPL_cleaned_data)

    daily_Alafia_slack_vector, \
    daily_Reservoir_slack_vector, \
    daily_TBC_slack_vector = getSWPermitViolations(AMPL_cleaned_data)

    # calculate reliability as the fraction of days without permit violations
    # and take count of longest stretch of consecutive days of violation
    # (if greater than 365, entire realization is a failure, which is a good
    # way to track firm yield on the supply under stationary demand realizations
    # but may not function well under transient conditions to track failure)
    violation_tracker = np.stack((daily_CWUP_overage_vector,
                                  daily_SCH_overage_vector,
                                  daily_BUD_overage_vector,
                                  daily_Alafia_slack_vector,
                                  daily_Reservoir_slack_vector,
                                  daily_TBC_slack_vector), axis = 1)
```

```python
    # total daily violations for the region
    daily_violation_sum = np.nansum(violation_tracker, axis = 1)

    # calculate basic ratio of days in violation to total days
    realization_level_of_service_reliability = sum(
            daily_violation_sum > 0) / len(daily_violation_sum)

    # length of all violation events
    import itertools
    length_of_events = [sum(1 for _ in group) \
                        for key, group in \
                        itertools.groupby(daily_violation_sum > 0) if key]

    # check if largest event is at least 365 days
    if len(length_of_events) > 0:
        if max(length_of_events) > 364:
            print('Realization in failure')

    # calculate number of failure events (14+ consec days of failure)
    realization_count_of_failure_events_vulnerability = len(
            [True for x in length_of_events if x > 13])

    return [realization_level_of_service_reliability,
            realization_count_of_failure_events_vulnerability]
```

GW permit violations

```python
def getGWPermitViolations(AMPL_cleaned_data):
    # names of GW permit violation variables (if tracked)
    Slack_Variable_Names = ['wup_mavg_pos__CWUP',
                            'wup_mavg_pos__SCH',
                            'wup_mavg_pos__BUD']

    # initialize vectors of NaN for each variable
    daily_overage_matrix = np.zeros([len(AMPL_cleaned_data),
                                     len(Slack_Variable_Names)])
    daily_overage_matrix[:] = np.nan

    # loop to collect data of interest
    for i in range(len(Slack_Variable_Names)):
        if Slack_Variable_Names[i] in AMPL_cleaned_data.columns:
            daily_overage_matrix[:,i] = AMPL_cleaned_data[Slack_Variable_Names[i]]
        else:
            daily_overage_matrix[:,i] = 0

    # rename for clarity and return
    daily_CWUP_overage_vector = daily_overage_matrix[:,0]
    daily_SCH_overage_vector  = daily_overage_matrix[:,1]
    daily_BUD_overage_vector  = daily_overage_matrix[:,2]

    return [daily_CWUP_overage_vector,
            daily_SCH_overage_vector,
            daily_BUD_overage_vector]
```

GW permit violations

Slack variables used

```python
def getGWPermitViolations(AMPL_cleaned_data):
    # names of GW permit violation variables (if tracked)
    Slack_Variable_Names = ['wup_mavg_pos__CWUP',
                            'wup_mavg_pos__SCH',
                            'wup_mavg_pos__BUD']

    # initialize vectors of NaN for each variable
    daily_overage_matrix = np.zeros([len(AMPL_cleaned_data),
                                     len(Slack_Variable_Names)])
    daily_overage_matrix[:] = np.nan

    # loop to collect data of interest
    for i in range(len(Slack_Variable_Names)):
        if Slack_Variable_Names[i] in AMPL_cleaned_data.columns:
            daily_overage_matrix[:,i] = AMPL_cleaned_data[Slack_Variable_Names[i]]
        else:
            daily_overage_matrix[:,i] = 0

    # rename for clarity and return
    daily_CWUP_overage_vector = daily_overage_matrix[:,0]
    daily_SCH_overage_vector  = daily_overage_matrix[:,1]
    daily_BUD_overage_vector  = daily_overage_matrix[:,2]

    return [daily_CWUP_overage_vector,
            daily_SCH_overage_vector,
            daily_BUD_overage_vector]
```

GW permit violations

```python
def getGWPermitViolations(AMPL_cleaned_data):
    # names of GW permit violation variables (if tracked)
    Slack_Variable_Names = ['wup_mavg_pos__CWUP',
                            'wup_mavg_pos__SCH',
                            'wup_mavg_pos__BUD']

    # initialize vectors of NaN for each variable
    daily_overage_matrix = np.zeros([len(AMPL_cleaned_data),
                                     len(Slack_Variable_Names)])
    daily_overage_matrix[:] = np.nan

    # loop to collect data of interest
    for i in range(len(Slack_Variable_Names)):
        if Slack_Variable_Names[i] in AMPL_cleaned_data.columns:
            daily_overage_matrix[:,i] = AMPL_cleaned_data[Slack_Variable_Names[i]]
        else:
            daily_overage_matrix[:,i] = 0

    # rename for clarity and return
    daily_CWUP_overage_vector = daily_overage_matrix[:,0]
    daily_SCH_overage_vector  = daily_overage_matrix[:,1]
    daily_BUD_overage_vector  = daily_overage_matrix[:,2]

    return [daily_CWUP_overage_vector,
            daily_SCH_overage_vector,
            daily_BUD_overage_vector]
```

GW permit violations

```python
def getGWPermitViolations(AMPL_cleaned_data):
    # names of GW permit violation variables (if tracked)
    Slack_Variable_Names = ['wup_mavg_pos__CWUP',
                            'wup_mavg_pos__SCH',
                            'wup_mavg_pos__BUD']

    # initialize vectors of NaN for each variable
    daily_overage_matrix = np.zeros([len(AMPL_cleaned_data),
                                     len(Slack_Variable_Names)])
    daily_overage_matrix[:] = np.nan

    # loop to collect data of interest
    for i in range(len(Slack_Variable_Names)):
        if Slack_Variable_Names[i] in AMPL_cleaned_data.columns:
            daily_overage_matrix[:,i] = AMPL_cleaned_data[Slack_Variable_Names[i]]
        else:
            daily_overage_matrix[:,i] = 0

    # rename for clarity and return
    daily_CWUP_overage_vector = daily_overage_matrix[:,0]
    daily_SCH_overage_vector  = daily_overage_matrix[:,1]
    daily_BUD_overage_vector  = daily_overage_matrix[:,2]

    return [daily_CWUP_overage_vector,
            daily_SCH_overage_vector,
            daily_BUD_overage_vector]
```

Returns vectors of daily at each location

calculateLevelOfService.py

Step 2: Calculate surface water violations (Alafia, Reservoir, TBC)

```
def calculateLevelOfService(AMPL_cleaned_data):
    # call "get" functions to pull data necessary to calculate objective
    # statistics for the realization
    daily_CWUP_overage_vector, \
    daily_SCH_overage_vector, \
    daily_BUD_overage_vector = getGWPermitViolations(AMPL_cleaned_data)

    daily_Alafia_slack_vector, \
    daily_Reservoir_slack_vector, \
    daily_TBC_slack_vector = getSWPermitViolations(AMPL_cleaned_data)

    # calculate reliability as the fraction of days without permit violations
    # and take count of longest stretch of consecutive days of violation
    # (if greater than 365, entire realization is a failure, which is a good
    # way to track firm yield on the supply under stationary demand realizations
    # but may not function well under transient conditions to track failure)
    violation_tracker = np.stack((daily_CWUP_overage_vector,
                                  daily_SCH_overage_vector,
                                  daily_BUD_overage_vector,
                                  daily_Alafia_slack_vector,
                                  daily_Reservoir_slack_vector,
                                  daily_TBC_slack_vector), axis = 1)
```

```
    # total daily violations for the region
    daily_violation_sum = np.nansum(violation_tracker, axis = 1)

    # calculate basic ratio of days in violation to total days
    realization_level_of_service_reliability = sum(
            daily_violation_sum > 0) / len(daily_violation_sum)

    # length of all violation events
    import itertools
    length_of_events = [sum(1 for _ in group) \
                        for key, group in \
                        itertools.groupby(daily_violation_sum > 0) if key]

    # check if largest event is at least 365 days
    if len(length_of_events) > 0:
        if max(length_of_events) > 364:
            print('Realization in failure')

    # calculate number of failure events (14+ consec days of failure)
    realization_count_of_failure_events_vulnerability = len(
            [True for x in length_of_events if x > 13])

    return [realization_level_of_service_reliability,
            realization_count_of_failure_events_vulnerability]
```

SW permit violations

```python
def getSWPermitViolations(AMPL_cleaned_data):
    # names of GW permit violation variables (if tracked)
    Slack_Variable_Names = ['ngw_slack__Alafia',
                            'ngw_slack__Reservoir',
                            'ngw_slack__TBC']

    # initialize matrix of NaN for variables
    daily_slack_matrix = np.zeros([len(AMPL_cleaned_data),
                                   len(Slack_Variable_Names)])
    daily_slack_matrix[:] = np.nan

    # loop to collect data of interest
    for i in range(len(Slack_Variable_Names)):
        if Slack_Variable_Names[i] in AMPL_cleaned_data.columns:
            daily_slack_matrix[:,i] = AMPL_cleaned_data[Slack_Variable_Names[i]]

    # rename for clarity and return
    daily_Alafia_slack_vector    = daily_slack_matrix[:,0]
    daily_Reservoir_slack_vector = daily_slack_matrix[:,1]
    daily_TBC_slack_vector       = daily_slack_matrix[:,2]

    return [daily_Alafia_slack_vector,
            daily_Reservoir_slack_vector,
            daily_TBC_slack_vector]
```

calculateLevelOfService.py

```python
def calculateLevelOfService(AMPL_cleaned_data):
    # call "get" functions to pull data necessary to calculate objective
    # statistics for the realization
    daily_CWUP_overage_vector, \
    daily_SCH_overage_vector, \
    daily_BUD_overage_vector = getGWPermitViolations(AMPL_cleaned_data)

    daily_Alafia_slack_vector, \
    daily_Reservoir_slack_vector, \
    daily_TBC_slack_vector = getSWPermitViolations(AMPL_cleaned_data)

    # calculate reliability as the fraction of days without permit violations
    # and take count of longest stretch of consecutive days of violation
    # (if greater than 365, entire realization is a failure, which is a good
    # way to track firm yield on the supply under stationary demand realizations
    # but may not function well under transient conditions to track failure)
    violation_tracker = np.stack((daily_CWUP_overage_vector,
                                  daily_SCH_overage_vector,
                                  daily_BUD_overage_vector,
                                  daily_Alafia_slack_vector,
                                  daily_Reservoir_slack_vector,
                                  daily_TBC_slack_vector), axis = 1)
```

```python
    # total daily violations for the region
    daily_violation_sum = np.nansum(violation_tracker, axis = 1)

    # calculate basic ratio of days in violation to total days
    realization_level_of_service_reliability = sum(
            daily_violation_sum > 0) / len(daily_violation_sum)

    # length of all violation events
    import itertools
    length_of_events = [sum(1 for _ in group) \
                        for key, group in \
                        itertools.groupby(daily_violation_sum > 0) if key]

    # check if largest event is at least 365 days
    if len(length_of_events) > 0:
        if max(length_of_events) > 364:
            print('Realization in failure')

    # calculate number of failure events (14+ consec days of failure)
    realization_count_of_failure_events_vulnerability = len(
            [True for x in length_of_events if x > 13])

    return [realization_level_of_service_reliability,
            realization_count_of_failure_events_vulnerability]
```

Step 3: Add all violations to a matrix

Day	CWUP	SCH	BUD	Alf	Res	TBC
1	0. 12	0	.2	1	0	0
…	…	…	…	…	…	…
7305	0	0	0	0	0	0

calculateLevelOfService.py

Step 4: Sum the days with any violation

```python
def calculateLevelOfService(AMPL_cleaned_data):
    # call "get" functions to pull data necessary to calculate objective
    # statistics for the realization
    daily_CWUP_overage_vector, \
    daily_SCH_overage_vector, \
    daily_BUD_overage_vector = getGWPermitViolations(AMPL_cleaned_data)

    daily_Alafia_slack_vector, \
    daily_Reservoir_slack_vector, \
    daily_TBC_slack_vector = getSWPermitViolations(AMPL_cleaned_data)

    # calculate reliability as the fraction of days without permit violations
    # and take count of longest stretch of consecutive days of violation
    # (if greater than 365, entire realization is a failure, which is a good
    # way to track firm yield on the supply under stationary demand realizations
    # but may not function well under transient conditions to track failure)
    violation_tracker = np.stack((daily_CWUP_overage_vector,
                                  daily_SCH_overage_vector,
                                  daily_BUD_overage_vector,
                                  daily_Alafia_slack_vector,
                                  daily_Reservoir_slack_vector,
                                  daily_TBC_slack_vector), axis = 1)
```

```python
    # total daily violations for the region
    daily_violation_sum = np.nansum(violation_tracker, axis = 1)

    # calculate basic ratio of days in violation to total days
    realization_level_of_service_reliability = sum(
            daily_violation_sum > 0) / len(daily_violation_sum)

    # length of all violation events
    import itertools
    length_of_events = [sum(1 for _ in group) \
                        for key, group in \
                        itertools.groupby(daily_violation_sum > 0) if key]

    # check if largest event is at least 365 days
    if len(length_of_events) > 0:
        if max(length_of_events) > 364:
            print('Realization in failure')

    # calculate number of failure events (14+ consec days of failure)
    realization_count_of_failure_events_vulnerability = len(
            [True for x in length_of_events if x > 13])

    return [realization_level_of_service_reliability,
            realization_count_of_failure_events_vulnerability]
```

Day	CWUP	SCH	BUD	Alf	Res	TBC	**Fail?**
1	0. 12	0	.2	1	0	0	**1**
…	…	…	…	…	…	…	
7305	0	0	0	0	0	0	**0**

calculateLevelOfService.py

Step 5: Calculate failure rate

```
def calculateLevelOfService(AMPL_cleaned_data):
    # call "get" functions to pull data necessary to calculate objective
    # statistics for the realization
    daily_CWUP_overage_vector, \
    daily_SCH_overage_vector, \
    daily_BUD_overage_vector = getGWPermitViolations(AMPL_cleaned_data)

    daily_Alafia_slack_vector, \
    daily_Reservoir_slack_vector, \
    daily_TBC_slack_vector = getSWPermitViolations(AMPL_cleaned_data)

    # calculate reliability as the fraction of days without permit violations
    # and take count of longest stretch of consecutive days of violation
    # (if greater than 365, entire realization is a failure, which is a good
    # way to track firm yield on the supply under stationary demand realizations
    # but may not function well under transient conditions to track failure)
    violation_tracker = np.stack((daily_CWUP_overage_vector,
                                  daily_SCH_overage_vector,
                                  daily_BUD_overage_vector,
                                  daily_Alafia_slack_vector,
                                  daily_Reservoir_slack_vector,
                                  daily_TBC_slack_vector), axis = 1)
```

```
    # total daily violations for the region
    daily_violation_sum = np.nansum(violation_tracker, axis = 1)

    # calculate basic ratio of days in violation to total days
    realization_level_of_service_reliability = sum(
            daily_violation_sum > 0) / len(daily_violation_sum)

    # length of all violation events
    import itertools
    length_of_events = [sum(1 for _ in group) \
                        for key, group in \
                        itertools.groupby(daily_violation_sum > 0) if key]

    # check if largest event is at least 365 days
    if len(length_of_events) > 0:
        if max(length_of_events) > 364:
            print('Realization in failure')

    # calculate number of failure events (14+ consec days of failure)
    realization_count_of_failure_events_vulnerability = len(
            [True for x in length_of_events if x > 13])

    return [realization_level_of_service_reliability,
            realization_count_of_failure_events_vulnerability]
```

Day	CWUP	SCH	BUD	Alf	Res	TBC	Fail?
1	0. 12	0	.2	1	0	0	**1**
…	…	…	…	…	…	…	
7305	0	0	0	0	0	0	**0**

$$\frac{\text{Sum(fail)}}{7305}$$

calculateLevelOfService.py

Step 6: Calculate length of failure periods

```python
def calculateLevelOfService(AMPL_cleaned_data):
    # call "get" functions to pull data necessary to calculate objective
    # statistics for the realization
    daily_CWUP_overage_vector, \
    daily_SCH_overage_vector, \
    daily_BUD_overage_vector = getGWPermitViolations(AMPL_cleaned_data)

    daily_Alafia_slack_vector, \
    daily_Reservoir_slack_vector, \
    daily_TBC_slack_vector = getSWPermitViolations(AMPL_cleaned_data)

    # calculate reliability as the fraction of days without permit violations
    # and take count of longest stretch of consecutive days of violation
    # (if greater than 365, entire realization is a failure, which is a good
    # way to track firm yield on the supply under stationary demand realizations
    # but may not function well under transient conditions to track failure)
    violation_tracker = np.stack((daily_CWUP_overage_vector,
                                  daily_SCH_overage_vector,
                                  daily_BUD_overage_vector,
                                  daily_Alafia_slack_vector,
                                  daily_Reservoir_slack_vector,
                                  daily_TBC_slack_vector), axis = 1)
```

```python
    # total daily violations for the region
    daily_violation_sum = np.nansum(violation_tracker, axis = 1)

    # calculate basic ratio of days in violation to total days
    realization_level_of_service_reliability = sum(
            daily_violation_sum > 0) / len(daily_violation_sum)

    # length of all violation events
    import itertools
    length_of_events = [sum(1 for _ in group) \
                        for key, group in \
                        itertools.groupby(daily_violation_sum > 0) if key]

    # check if largest event is at least 365 days
    if len(length_of_events) > 0:
        if max(length_of_events) > 364:
            print('Realization in failure')

    # calculate number of failure events (14+ consec days of failure)
    realization_count_of_failure_events_vulnerability = len(
            [True for x in length_of_events if x > 13])

    return [realization_level_of_service_reliability,
            realization_count_of_failure_events_vulnerability]
```

calculateLevelOfService.py

Step 7: calculate number of 14+ day failure periods

```python
def calculateLevelOfService(AMPL_cleaned_data):
    # call "get" functions to pull data necessary to calculate objective
    # statistics for the realization
    daily_CWUP_overage_vector, \
    daily_SCH_overage_vector, \
    daily_BUD_overage_vector = getGWPermitViolations(AMPL_cleaned_data)

    daily_Alafia_slack_vector, \
    daily_Reservoir_slack_vector, \
    daily_TBC_slack_vector = getSWPermitViolations(AMPL_cleaned_data)

    # calculate reliability as the fraction of days without permit violations
    # and take count of longest stretch of consecutive days of violation
    # (if greater than 365, entire realization is a failure, which is a good
    # way to track firm yield on the supply under stationary demand realizations
    # but may not function well under transient conditions to track failure)
    violation_tracker = np.stack((daily_CWUP_overage_vector,
                                  daily_SCH_overage_vector,
                                  daily_BUD_overage_vector,
                                  daily_Alafia_slack_vector,
                                  daily_Reservoir_slack_vector,
                                  daily_TBC_slack_vector), axis = 1)
```

```python
    # total daily violations for the region
    daily_violation_sum = np.nansum(violation_tracker, axis = 1)

    # calculate basic ratio of days in violation to total days
    realization_level_of_service_reliability = sum(
            daily_violation_sum > 0) / len(daily_violation_sum)

    # length of all violation events
    import itertools
    length_of_events = [sum(1 for _ in group) \
                        for key, group in \
                        itertools.groupby(daily_violation_sum > 0) if key]

    # check if largest event is at least 365 days
    if len(length_of_events) > 0:
        if max(length_of_events) > 364:
            print('Realization in failure')

    # calculate number of failure events (14+ consec days of failure)
    realization_count_of_failure_events_vulnerability = len(
            [True for x in length_of_events if x > 13])

    return [realization_level_of_service_reliability,
            realization_count_of_failure_events_vulnerability]
```

time_varying_objectives.py

```python
def plotLOSQuantiles(n_rel, figureName):

    # load each cleaned AMPL outfile and store in a list
    all_ampl = list()

    for rel in range(1,n_rel):
        if rel < 10:
            num_str = '000'+ str(rel)
        elif rel < 100:
            num_str = '00' + str(rel)
        else:
            num_str = '0' + str(rel)
        realization_data = pd.read_csv('Cleaned_zip/cleaned/ampl_' + num_str +
                                       '.csv')#, usecols=AMPL_cols)
        realization_data.fillna(0, inplace=True)

        all_ampl.append(realization_data)

    # Calculate LOS metrics for each year (missing leap years, fix later?)
    annual_LOS_rel = np.zeros([n_rel, 20])
    annual_LOS_vul = np.zeros([n_rel, 20])

    # loop through each realization
    for rel in range(0, n_rel-1):
        current_rel = all_ampl[rel]

        for day in np.linspace(0, 6935, 20):
            day = int(day)
            year = int(day/365)
            current_year = current_rel.iloc[day:day+365]
            annual_LOS_rel[rel, year], annual_LOS_vul[rel, year] = calculateLevelOfService(current_year)


    # Calculate percentiles across realizations
    LOS_rel = np.zeros([20,100])
    LOS_vul = np.zeros([20,100])
    # Extract 90th percentile from each LOS
    for year in range(0,20):
        for p in range(0,100):
            LOS_rel[year,p] = np.percentile(annual_LOS_rel[:,year], (p+1))
            LOS_vul[year,p] = np.percentile(annual_LOS_vul[:,year], (p+1))
```

```python
    # plot time varying distribution
    cmap = matplotlib.cm.get_cmap("RdBu_r")
    fig = plt.figure(figsize=(8,6))
    ax = fig.add_subplot(1,1,1)
    for i in np.linspace(5,95,19):
        i = int(i)
        ax.fill_between(range(2020,2040), LOS_rel[:,i-5], LOS_rel[:,i],
                        color=cm.RdBu_r((i-1)/100.0), alpha=0.75, edgecolor='none')
    ax.set_xticks(range(2020,2040))
    ax.set_xticklabels(range(2020,2040), rotation='vertical')
    ax.set_xlim([2020,2039])
    plt.xlabel('years')
    plt.ylabel('Annual failure rate')

    plt.savefig(figureName, bbox_inches= 'tight')

    return
```

time_varying_objectives.py

```python
def plotLOSQuantiles(n_rel, figureName):

    # load each cleaned AMPL outfile and store in a list
    all_ampl = list()

    for rel in range(1,n_rel):
        if rel < 10:
            num_str = '000'+ str(rel)
        elif rel < 100:
            num_str = '00' + str(rel)
        else:
            num_str = '0' + str(rel)
        realization_data = pd.read_csv('Cleaned_zip/cleaned/ampl_' + num_str +
                                       '.csv')#, usecols=AMPL_cols)
        realization_data.fillna(0, inplace=True)

        all_ampl.append(realization_data)

    # Calculate LOS metrics for each year (missing leap years, fix later?)
    annual_LOS_rel = np.zeros([n_rel, 20])
    annual_LOS_vul = np.zeros([n_rel, 20])

    # loop through each realization
    for rel in range(0, n_rel-1):
        current_rel = all_ampl[rel]

        for day in np.linspace(0, 6935, 20):
            day = int(day)
            year = int(day/365)
            current_year = current_rel.iloc[day:day+365]
            annual_LOS_rel[rel, year], annual_LOS_vul[rel, year] = calculateLevelOfService(current_year


    # Calculate percentiles across realizations
    LOS_rel = np.zeros([20,100])
    LOS_vul = np.zeros([20,100])
    # Extract 90th percentile from each LOS
    for year in range(0,20):
        for p in range(0,100):
            LOS_rel[year,p] = np.percentile(annual_LOS_rel[:,year], (p+1))
            LOS_vul[year,p] = np.percentile(annual_LOS_vul[:,year], (p+1))
```

```python
    # plot time varying distribution
    cmap = matplotlib.cm.get_cmap("RdBu_r")
    fig = plt.figure(figsize=(8,6))
    ax = fig.add_subplot(1,1,1)
    for i in np.linspace(5,95,19):
        i = int(i)
        ax.fill_between(range(2020,2040), LOS_rel[:,i-5], LOS_rel[:,i],
                        color=cm.RdBu_r((i-1)/100.0), alpha=0.75, edgecolor='none')
    ax.set_xticks(range(2020,2040))
    ax.set_xticklabels(range(2020,2040), rotation='vertical')
    ax.set_xlim([2020,2039])
    plt.xlabel('years')
    plt.ylabel('Annual failure rate')

    plt.savefig(figureName, bbox_inches= 'tight')

    return
```

Step 3: Calculate percentiles of across realizations

Percentiles

Year	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
1	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02
2	0.00	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02
3	0.00	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02
4	0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02
5	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02
6	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03	0.03
7	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02
8	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02
9	0.00	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02
10	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02
11	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02
12	0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02
13	0.00	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03
14	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
15	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02
16	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03
17	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03
18	0.00	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.03
19	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03
20	0.00	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.03	0.03

…

Year	90%	91%	92%	93%	94%	95%	96%	97%	98%	99%	100%
1	0.10	0.10	0.10	0.11	0.11	0.12	0.12	0.12	0.12	0.13	0.15
2	0.13	0.13	0.13	0.14	0.15	0.15	0.16	0.16	0.17	0.29	$\boxed{0.32}$
3	0.13	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.18	[illegible]	0.36
4	0.13	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.16	0.16	0.17
5	0.14	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.18	0.21	0.22
6	0.12	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.26	0.30	0.31
7	0.14	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.19	0.22	0.29
8	0.13	0.14	0.14	0.14	0.16	0.18	0.19	0.32	0.32	0.36	0.40
9	0.12	0.13	0.14	0.15	0.16	0.18	0.18	0.20	0.23	0.37	0.52
10	0.12	0.12	0.13	0.15	0.15	0.16	0.17	0.19	0.32	0.43	0.46
11	0.18	0.19	0.19	0.20	0.20	0.22	0.23	0.26	0.27	0.31	0.47
12	0.19	0.19	0.19	0.20	0.23	0.24	0.25	0.27	0.27	0.30	0.37
13	0.18	0.19	0.20	0.20	0.20	0.23	0.24	0.31	0.33	0.42	0.45
14	0.21	0.21	0.23	0.25	0.25	0.26	0.27	0.33	0.34	0.35	0.44
15	0.21	0.22	0.23	0.23	0.24	0.24	0.28	0.32	0.33	0.35	0.37
16	0.21	0.22	0.22	0.22	0.23	0.25	0.27	0.28	0.37	0.38	0.47
17	0.22	0.22	0.24	0.25	0.25	0.25	0.26	0.28	0.31	0.36	0.38
18	0.22	0.22	0.23	0.23	0.24	0.25	0.26	0.28	0.29	0.42	0.48
19	0.22	0.22	0.23	0.23	0.24	0.25	0.25	0.27	0.34	0.46	0.51
20	0.21	0.21	0.21	0.22	0.22	0.23	0.25	0.26	0.31	0.34	0.59

highest failure rate across 1000 realizations in **year 2**

Percentiles

Year	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%	…	90%	91%	92%	93%	94%	95%	96%	97%	98%	99%	100%
1	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.10	0.10	0.10	0.11	0.11	0.12	0.12	0.12	0.12	0.13	0.15
2	0.00	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02		0.13	0.13	0.13	0.14	0.15	0.15	0.16	0.16	0.17	0.29	0.32
3	0.00	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02		0.13	0.13	0.13	0.13	0.14	0.14	0.15	0.16	0.18	0.18	0.36
4	0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.16	0.16	0.16	0.17
5	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02		0.14	0.14	0.15	0.16	0.16	0.17	0.17	0.17	0.18	0.21	0.22
6	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03	0.03		0.12	0.13	0.14	0.14	0.14	0.15	0.18	0.20	0.26	0.30	0.31
7	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02		0.14	0.15	0.15	0.15	0.15	0.18	0.18	0.18	0.19	0.22	0.29
8	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02		0.13	0.14	0.14	0.14	0.16	0.18	0.19	0.32	0.32	0.36	0.40
9	0.00	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02		0.12	0.13	0.14	0.15	0.16	0.18	0.18	0.20	0.23	0.37	0.52
10	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	…	0.12	0.12	0.13	0.15	0.15	0.16	0.17	0.19	0.32	0.43	0.46
11	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.18	0.19	0.19	0.20	0.20	0.22	0.23	0.26	0.27	0.31	0.47
12	0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02		0.19	0.19	0.19	0.20	0.23	0.24	0.25	0.27	0.27	0.30	0.37
13	0.00	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03		0.18	0.19	0.20	0.20	0.20	0.23	0.24	0.31	0.33	0.42	0.45
14	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02		0.21	0.21	0.23	0.25	0.25	0.26	0.27	0.33	0.34	0.35	0.44
15	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02		0.21	0.22	0.23	0.23	0.24	0.24	0.28	0.32	0.33	0.35	0.37
16	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03		0.21	0.22	0.22	0.22	0.23	0.25	0.27	0.28	0.37	0.38	0.47
17	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03		0.22	0.22	0.24	0.25	0.25	0.25	0.26	0.28	0.31	0.36	0.38
18	0.00	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.03		0.22	0.22	0.23	0.23	0.24	0.25	0.26	0.28	0.29	0.42	0.48
19	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03		0.22	0.22	0.23	0.23	0.24	0.25	0.25	0.27	0.34	0.46	0.51
20	0.00	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.03	0.03		0.21	0.21	0.21	0.22	0.22	0.23	0.25	0.26	0.31	0.34	0.59

time_varying_objectives.py

```python
def plotLOSQuantiles(n_rel, figureName):

    # load each cleaned AMPL outfile and store in a list
    all_ampl = list()

    for rel in range(1,n_rel):
        if rel < 10:
            num_str = '000'+ str(rel)
        elif rel < 100:
            num_str = '00' + str(rel)
        else:
            num_str = '0' + str(rel)
        realization_data = pd.read_csv('Cleaned_zip/cleaned/ampl_' + num_str +
                                       '.csv')#, usecols=AMPL_cols)
        realization_data.fillna(0, inplace=True)

        all_ampl.append(realization_data)

    # Calculate LOS metrics for each year (missing leap years, fix later?)
    annual_LOS_rel = np.zeros([n_rel, 20])
    annual_LOS_vul = np.zeros([n_rel, 20])

    # loop through each realization
    for rel in range(0, n_rel-1):
        current_rel = all_ampl[rel]

        for day in np.linspace(0, 6935, 20):
            day = int(day)
            year = int(day/365)
            current_year = current_rel.iloc[day:day+365]
            annual_LOS_rel[rel, year], annual_LOS_vul[rel, year] = calculateLevelOfService(current_year)


    # Calculate percentiles across realizations
    LOS_rel = np.zeros([20,100])
    LOS_vul = np.zeros([20,100])
    # Extract 90th percentile from each LOS
    for year in range(0,20):
        for p in range(0,100):
            LOS_rel[year,p] = np.percentile(annual_LOS_rel[:,year], (p+1))
            LOS_vul[year,p] = np.percentile(annual_LOS_vul[:,year], (p+1))
```

```python
    # plot time varying distribution
    cmap = matplotlib.cm.get_cmap("RdBu_r")
    fig = plt.figure(figsize=(8,6))
    ax = fig.add_subplot(1,1,1)
    for i in np.linspace(5,95,19):
        i = int(i)
        ax.fill_between(range(2020,2040), LOS_rel[:,i-5], LOS_rel[:,i],
                        color=cm.RdBu_r((i-1)/100.0), alpha=0.75, edgecolor='none')
    ax.set_xticks(range(2020,2040))
    ax.set_xticklabels(range(2020,2040), rotation='vertical')
    ax.set_xlim([2020,2039])
    plt.xlabel('years')
    plt.ylabel('Annual failure rate')

    plt.savefig(figureName, bbox_inches= 'tight')

    return
```

Step 4: Plot

Final product