Popping the Kernel

Using Constrained Optimization to Estimate the Values behind a Kernel Estimation

By Aneesh Sandhir

What is Kernel Estimation?

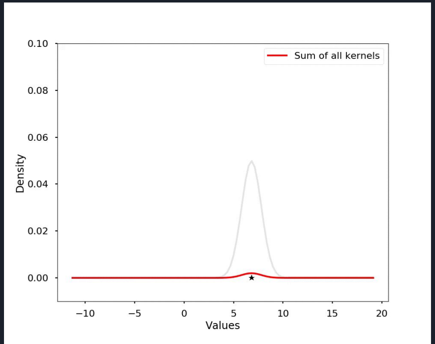
- Kernel Estimation is a transformation of a dataset through the application of a distribution, the kernel

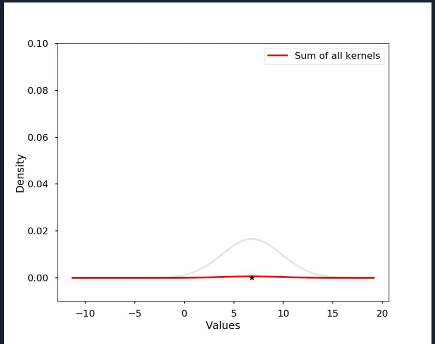
- Kernels are defined by their shape which determines the properties of the resulting transform

- A kernel’s width is its most important attribute

- Kernels can be useful tools to

  - Investigate spatial or cyclic patterns
  - Remove noise
  - Develop a probability density function





https://qingkaikong.blogspot.com/2018/05/kernel-density-estimation-animation.html

Why Undo a Kernel Estimation?

- The kernel obfuscates the data
- The kernel applied to the data can constrain your analysis
- There is almost no way to apply a different kernel to the data without the raw data itself
    - Only if the new kernel's size is a multiple of the kernel already applied to the data, this can be very limiting

How to Undo a Kernel Estimation?

- Each output of a Kernel Estimation can be viewed as its own linear equation
- And with many of the linear equations sharing variables, the entire transformation can be viewed as the following system of linear equations

$$A \times x = b$$

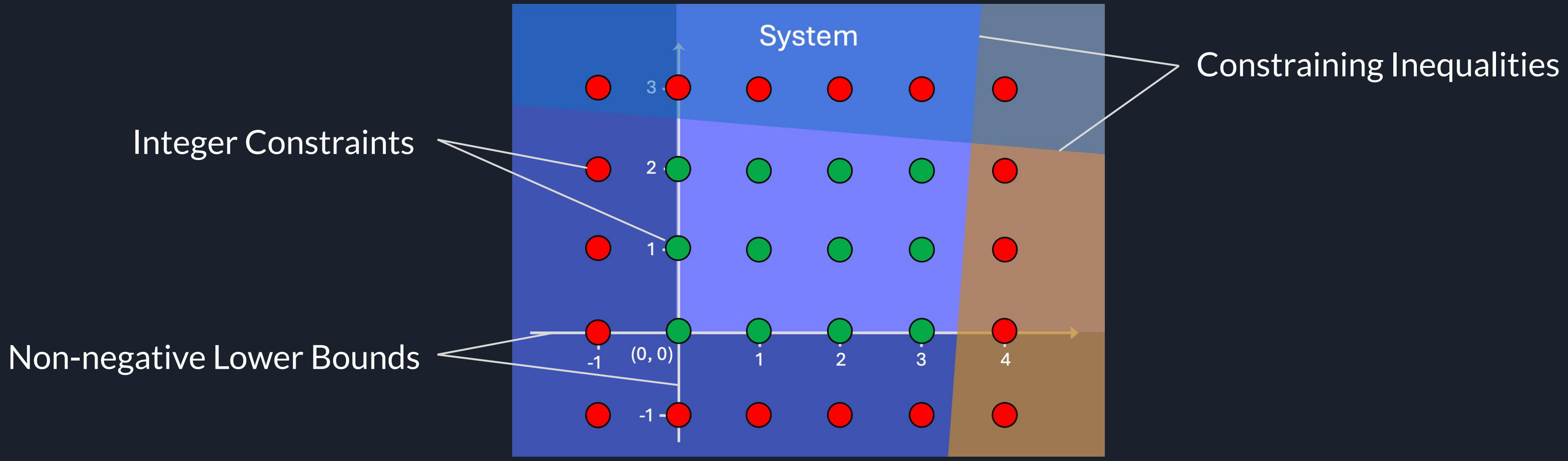
The coefficient matrix constructed from the kernel

Every output from the kernel estimation

Each underlying data point

How to Undo a Kernel Estimation? Continued

- Due to the nature of kernel density estimations, the system will have more variables than constraining equations, this means there are infinitely many solutions to the system
- This solution space can be reigned in by further constraining the system
- These constraints can take the form of inequalities, setting upper and lower bounds on the variables, or number classification, either continuous, integer, or binary



Where Constrained Optimization Fits into the Data Engineer’s Toolbox?

Data Transformation	**Action**	**Reaction**
Row Wise	Union	Filter
Column Wise	Join	Subset
Reshaping	Pivot	Melt
Accumulation	Cumulative Sum	Lag
Rolling Aggregations	**Kernel Estimation**	**Constrained Optimization**

One Dimensional Example:
Time Series - CDC Overdose Data

Vital Statistics Rapid Release Drug Overdose Data

- The VSRR data tracks the mortality statistics indexed by region, month and drug, spanning from 2015 to 2022
- The mortality statistics are recorded as the sum of the current month's death count and the death count of the preceding 11 months
- This hides some of the month-to-month variation and makes it easier to identify a general trend but obfuscates the underlying data and constrains the analysi through a 12-month window
- Each month's exact death count can't be determined, they can be estimated through constrained optimization

Developing the System of Equations

- A 12-month rolling sum is simply a uniform kernel with a width of 12 and a height of 1
- Developing the system starts by representing a given month as i, its 12-month rolling death tally Σ_i, the death tally for that month x_i, and the death tally for the preceding 11 months as $x_{i-11} \cdots x_i$ will yield the following constraining equation

$$x_{i-11} + x_{i-10} + x_{i-9} + x_{i-8} + x_{i-7} + x_{i-6} + x_{i-5} + x_{i-4} + x_{i-3} + x_{i-2} + x_{i-1} + x_i = \Sigma_i$$

- The next month can be represented as $i+1$, its 12-month rolling death tally as Σ_{i+1}, the death tally for that month as x_{i+1} and will yield the following constraining equation

$$x_{i-10} + x_{i-9} + x_{i-8} + x_{i-7} + x_{i-6} + x_{i-5} + x_{i-4} + x_{i-3} + x_{i-2} + x_{i-1} + x_i + x_{i+1} = \Sigma_{i+1}$$

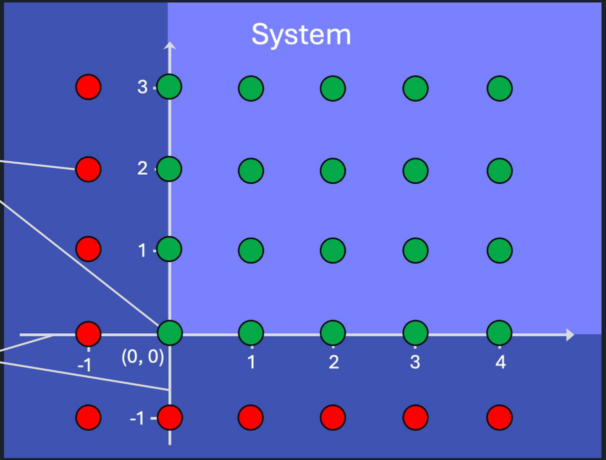
Constraining the System

- The system has 11 more variables than constraining equations
- Knowing the variables represent overdose deaths the solution space can be constrained to a finite set

- These additional constraints can be implemented by first confining the solution space to integers
- Followed by applying a constraining inequality where none of the elements in the variable matrix can be less than 0

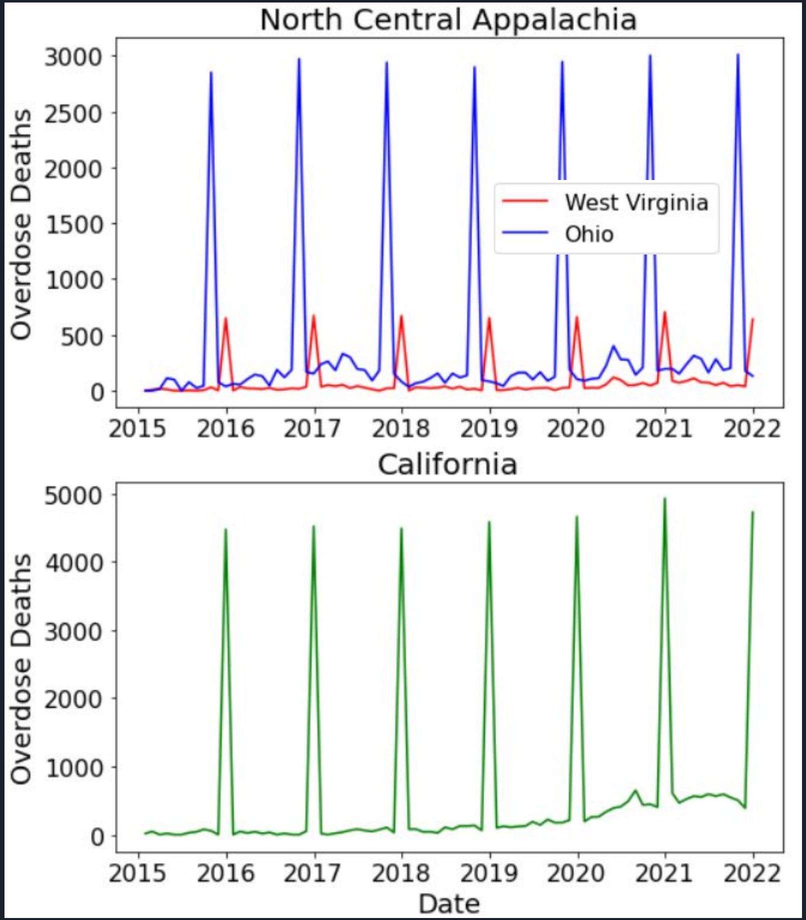
Integer Constraints

Non-negative Lower Bounds



Initial Solution is Unrealistic

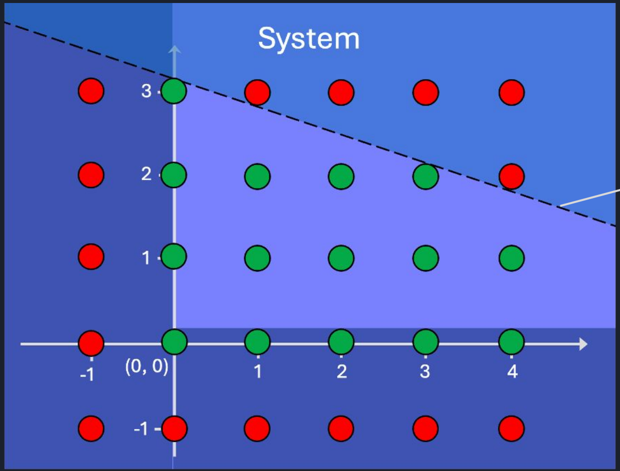
- Solving this system via optimization yields the least entropic, most volatile solution with an overwhelming majority of deaths occuring one month a year
- In all likelihood, the system does not behave this way



Further Constraining the System

- To get a less volatile solution, the system can be further constrained by applying another constraining inequality where none of the elements in the variable matrix can be exceed a given value

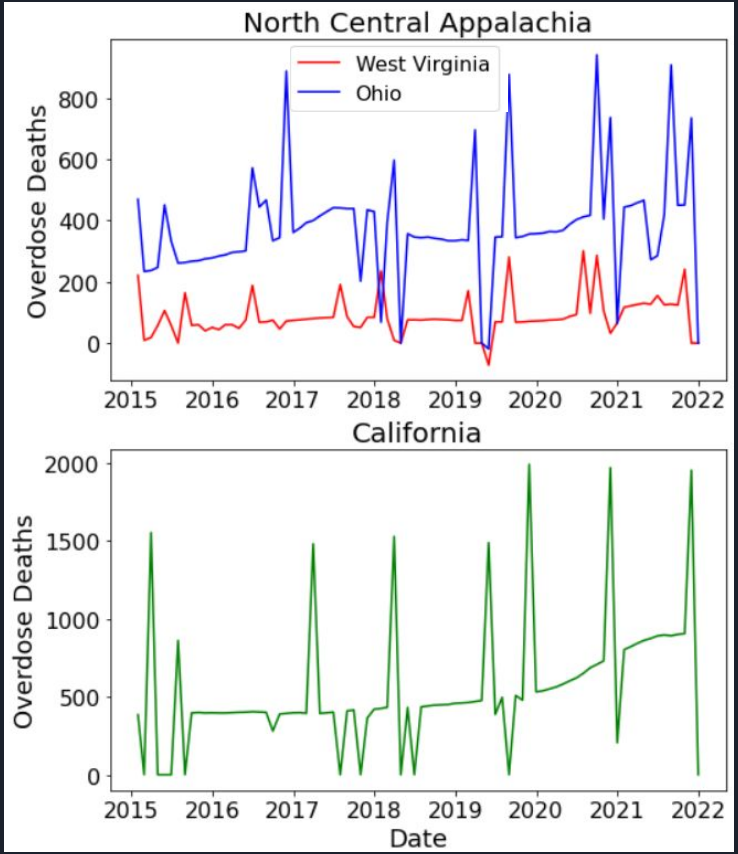
- Here it is implemented as a maximum percentage deviation **above** the 12-month moving average, also referred to as a smoothing or volatility factor



Dynamic Upper Bound

Low Volatility Factor - Overconstrained System

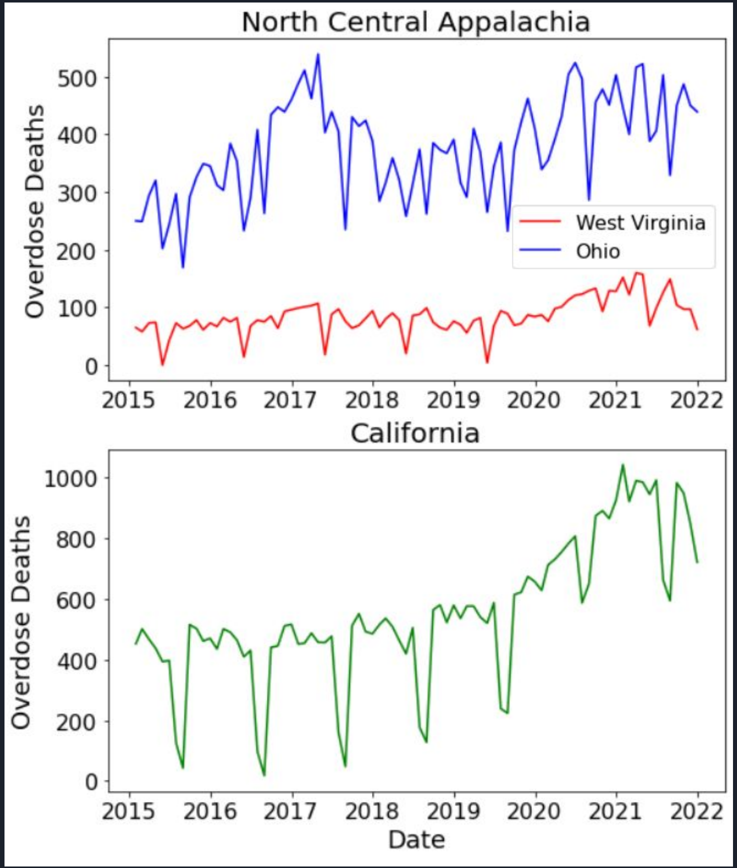
- If the volatility factor is too low, tightly tying the monthly values to the moving average, the system will be overly constrained and no solution can satisfy all the constraints put on the system



Optimal Volatility Factor

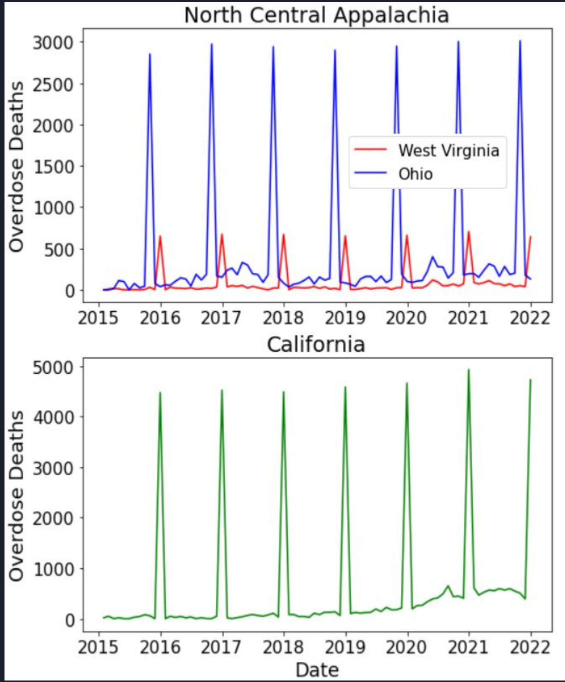
- If the volatility factor is too high the resulting solution will be artificially cyclical and very volatile, as was previously

- Selecting the right volatility factor depends on the ratio of unknown values to the constraining equations, the magnitude of the values in the series and can only be identified via trial and error

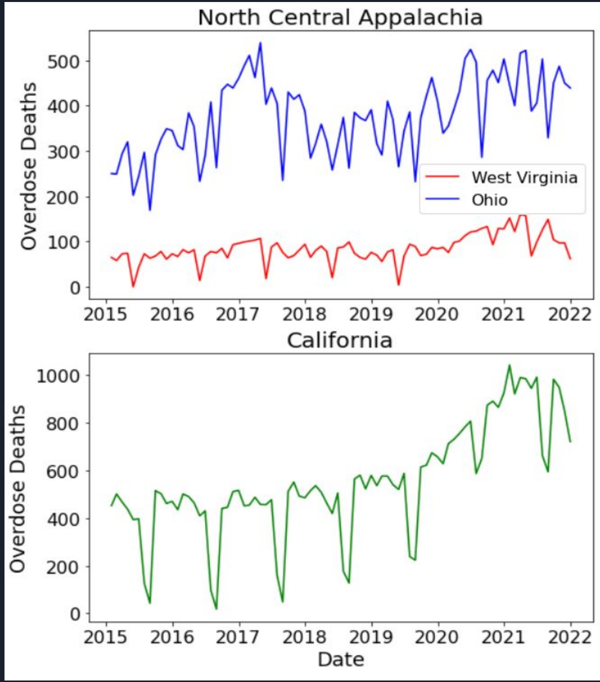


Comparison of Constraints

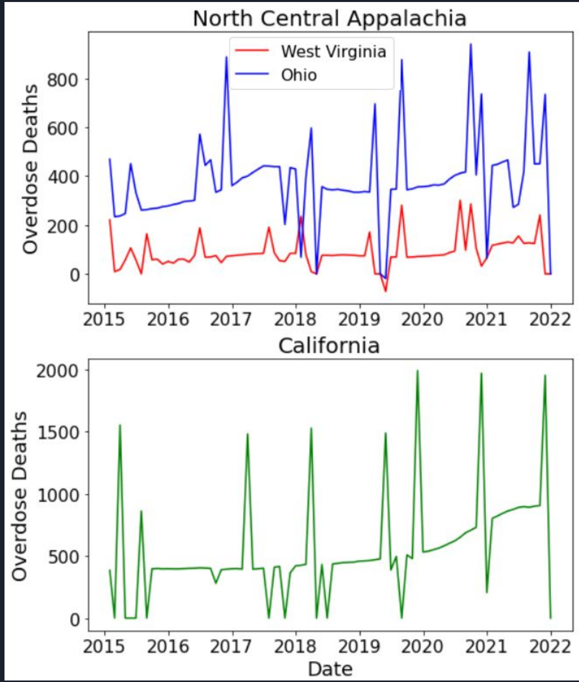
Under Constrained



Optimal Volatility Factor



Overly Constrained

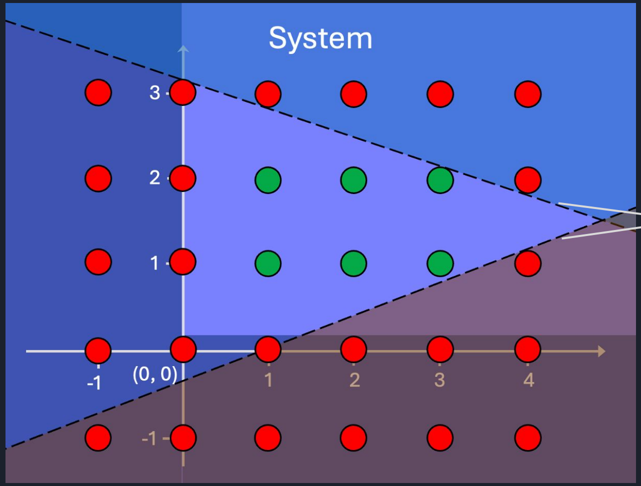


Additional Constraints - Dynamic Lower Bound

- Applying a dynamic upper bound to the variables results in a system that behaves a lot more realistic, but there are still drastic sharp rises and drops that appear at almost the same time every year

- Instead of applying a static lower bound of 0 to all the variables, the lower bound can be made dynamic as well

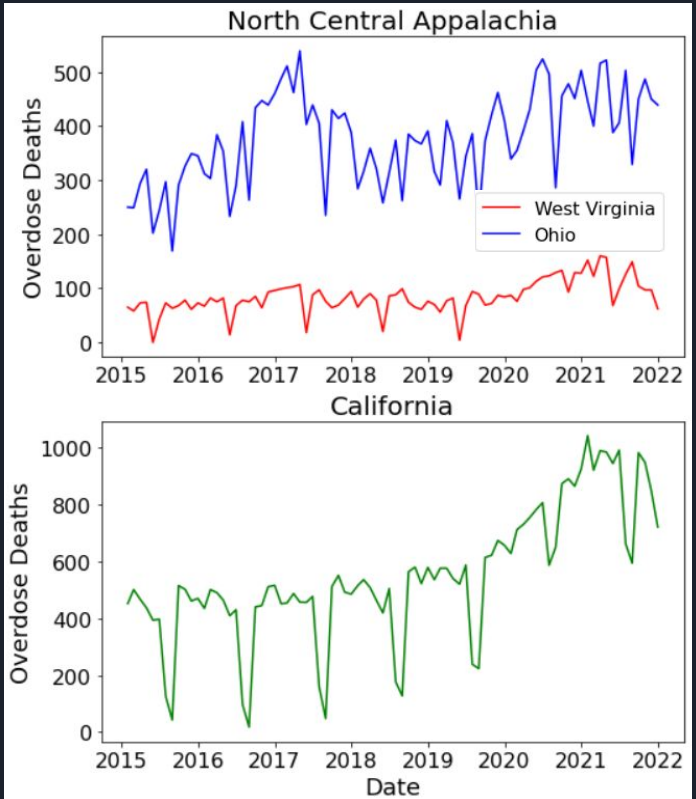
- Using the same volatility factor as the upper bound, the variables can have a maximum percentage deviation **below** the 12-month moving average



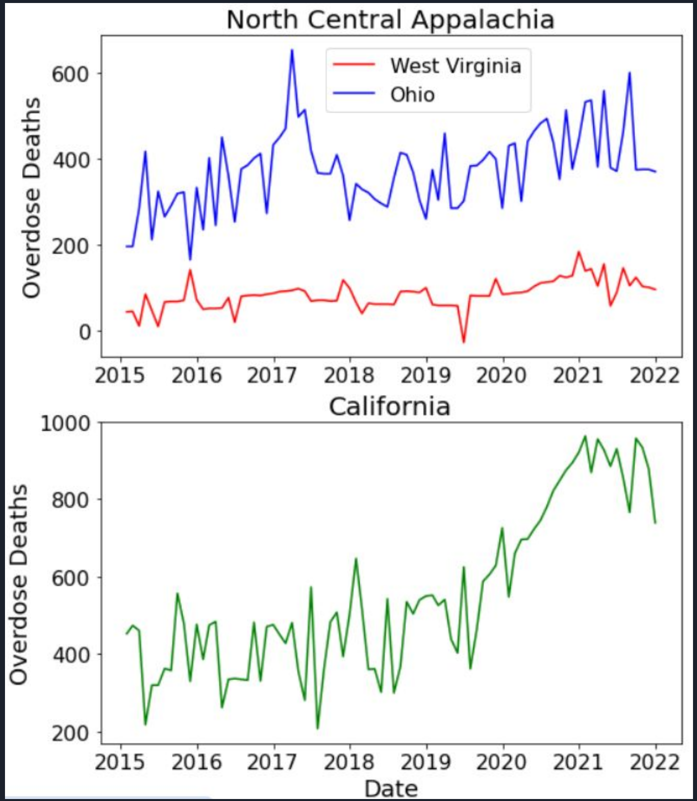
Dynamic Upper & Lower Bounds

Comparison of Constraints on the Lower Bound

Non-Negative Lower Bound

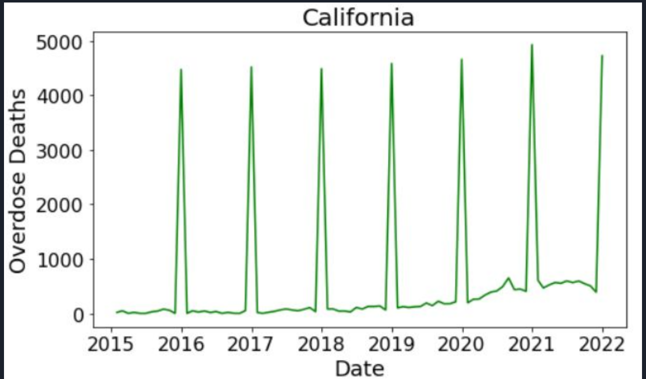


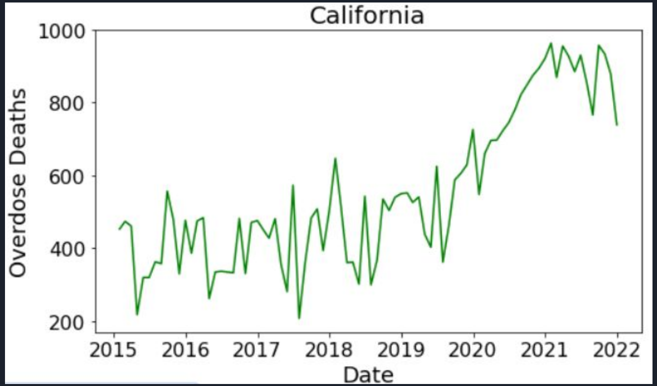
Dynamically Constrained Lower Bound



Conclusions

- By fine tuning the constraints applied to the system and identifying optimal volatility factor, we found a good estimate of the underlying data





- Transformed 12-month rolling sum into month by month data

- This monthly overdose data was used as as input into a descriptive model

- Models that used monthly overdose data outperformed the models which used rolling sum data

Initial Python Implementation

```python

#Import dependencies
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
import pulp
import time
import matplotlib.pyplot as plt

#Import VSRR data
...

#Create sliding window function
def sliding_windows(kernel, num_constraining_eqs):
    kernel = np.asarray(kernel)
    p = np.zeros(num_constraining_eqs-1,dtype=kernel.dtype)
    b = np.concatenate((p,kernel,p))
    s = b.strides[0]
    strided = np.lib.stride_tricks.as_strided
    return strided(b[num_constraining_eqs-1:], shape=(num_constraining_eqs,len(kernel)+num_constraining_eqs-1), strides=(-s,s

```

Initial Python Implementation Continued

```
period = 12
regions = death_df["Region"].unique()
indicies = ["State", "FIPS", "Region", "Year", "Month", "Start Date", "End Date", "Percent with drugs specified"]
targets = [element for element in death_df.columns.tolist() if element not in indicies]

monthly_deaths_df = pd.DataFrame()
for region in regions:
    series = death_df[death_df["Region"] == region].sort_values(by='End Date')
    for target in targets:
        sol = series[target].dropna().to_numpy()
        if (len(sol) < 2):
            marginal_values = np.empty(len(series.index))
            marginal_values[:] = np.nan
            series["Estimated Monthly Marginal " + target] = marginal_values
            continue
        coef = sliding_windows(np.ones(period), len(sol))
        #pulp Linear Algebra Solver yeilds a solution not guaranteed to be the correct solution but all the values
        #it produces will be non-negative, discrete and yeild the values in the dataset when summed across a 12 month window
        mod = pulp.LpProblem(region.replace(" ", "_") + ":" +  target.replace(" ", "_"))

        #set up contraining inequalities
        #constrain monthly death tally values to integers greater than or equal to 0
        vars = pulp.LpVariable.dicts('x', range(len(coef[0])), lowBound=0, cat='Integer')

        #set up contraining equations
        for row, rhs in zip(coef, sol):
            mod += sum([row[i]*vars[i] for i in range(len(row))]) == rhs

        mod.solve()

        #currently discarding the leading 11 values which dont have corresponding 12 month rolling sums
        #could possibly rewrite this section to keep them with null values for the 12 month rolling sums in the future
        marginal_values = [vars[i].value() for i in range(len(coef[0]))][(period -1):]
        series["Estimated Monthly Marginal " + target] = np.pad(marginal_values, (len(series.index)-len(marginal_values),0),

    monthly_deaths_df = pd.concat([monthly_deaths_df, series])
```

Python Implementation of More Constraints

```python
period = 12
volatility_factor = 1.2 # must be between 1 and period
regions = death_df["Region"].unique()
indicies = ["State", "FIPS", "Region", "Year", "Month", "Start Date", "End Date", "Percent with drugs specified"]
targets = [element for element in death_df.columns.tolist() if element not in indicies]

monthly_deaths_df = pd.DataFrame()
for region in regions:
    series = death_df[death_df["Region"] == region].sort_values(by='End Date')
    for target in targets:
        sol = series[target].dropna().to_numpy()
        if (len(sol) < 2):
            marginal_values = np.empty(len(series.index))
            marginal_values[:] = np.nan
            series["Estimated Monthly Marginal " + target] = marginal_values
            continue
        coef = sliding_windows(np.ones(period), len(sol))
        #pulp Linear Algebra Solver yeilds a solution not guaranteed to be the correct solution but all the values
        #it produces will be non-negative, discrete and yeild the values in the dataset when summed across a 12 month window
        mod = pulp.LpProblem(region.replace(" ", "_") + ":" +  target.replace(" ", "_"))

        #set up contraining inequalities
        #constrain monthly death tally values to integers greater than or equal to 0
        vars = pulp.LpVariable.dicts('x', range(len(coef[0])), lowBound=0, cat='Integer')

        #constrain monthly death tally values to be less than n% greater than themoving average
        upBound_vector = np.around(((volatility_factor/period)*np.pad(sol, (period-1,0), 'constant', constant_values=sol[0])
        for pointer in vars.keys():
            vars[pointer].upBound = upBound_vector[pointer]

        #set up contraining equations
        for row, rhs in zip(coef, sol):
            mod += sum([row[i]*vars[i] for i in range(len(row))]) == rhs

        mod.solve()
```

Python Implementation of More Refined Constraints

```python
volatility_factor = 1.2 # must be between 1 and period
regions = death_df["Region"].unique()
indicies = ["State", "FIPS", "Region", "Year", "Month", "Start Date", "End Date", "Percent with drugs specified"]
targets = [element for element in death_df.columns.tolist() if element not in indicies]

monthly_deaths_df = pd.DataFrame()
for region in regions:
    series = death_df[death_df["Region"] == region].sort_values(by='End Date')
    for target in targets:
        sol = series[target].dropna().to_numpy()
        if (len(sol) < 2):
            marginal_values = np.empty(len(series.index))
            marginal_values[:] = np.nan
            series["Estimated Monthly Marginal " + target] = marginal_values
            continue
        coef = sliding_windows(np.ones(period), len(sol))
        #pulp Linear Algebra Solver yeilds a solution not guaranteed to be the correct solution but all the values
        #it produces will be non-negative, discrete and yeild the values in the dataset when summed across a 12 month window
        mod = pulp.LpProblem(region.replace(" ", "_") + ":" +  target.replace(" ", "_"))

        #set up contraining inequalities
        #constrain monthly death tally values to integers greater than or equal to 0
        vars = pulp.LpVariable.dicts('x', range(len(coef[0])), lowBound=0, cat='Integer')

        #constrain monthly death tally values to be less than n% greater than themoving average
        lowBound_vector = np.around(((1/(volatility_factor*period))*np.pad(sol, (period-1,0), 'constant', constant_values=sol
        upBound_vector = np.around(((volatility_factor/period)*np.pad(sol, (period-1,0), 'constant', constant_values=sol[0])
        for pointer in vars.keys():
            vars[pointer].lowBound = lowBound_vector[pointer]
            vars[pointer].upBound = upBound_vector[pointer]

        #set up contraining equations
        for row, rhs in zip(coef, sol):
            mod += sum([row[i]*vars[i] for i in range(len(row))]) == rhs

        mod.solve()
```

Developing the System of Equations

- A 12-month rolling sum is simply a uniform kernel with a width of 12 and a height of 1
- Developing the system starts by representing a given month as i, its 12-month rolling death tally Σ_i, the death tally for that month x_i, and the death tally for the preceding 11 months as $x_{i-11} \cdots x_i$ will yield the following constraining equation

$$x_{i-11} + x_{i-10} + x_{i-9} + x_{i-8} + x_{i-7} + x_{i-6} + x_{i-5} + x_{i-4} + x_{i-3} + x_{i-2} + x_{i-1} + x_i = \Sigma_i$$

- The next month can be represented as $i+1$, its 12-month rolling death tally as Σ_{i+1}, the death tally for that month as x_{i+1} and will yield the following constraining equation

$$x_{i-10} + x_{i-9} + x_{i-8} + x_{i-7} + x_{i-6} + x_{i-5} + x_{i-4} + x_{i-3} + x_{i-2} + x_{i-1} + x_i + x_{i+1} = \Sigma_{i+1}$$