The Probability of Electoral Misalignment

Computing the probability that one candidate wins the electoral college while losing the popular vote for the 2020 election

Contents

1 Introduction

In American politics, it is not common for a presidential candidate to win the Electoral College vote while losing the popular vote, but if it happens one can be sure that there will be a lot of soul-searching commentary on this fairly intentional aspect of how the country selects its executive leader.

Reviewing the history of the Electoral College, it is apparent that the majority will of the people was something the Framers of the Constitution were deeply suspicious of:

> Direct election was rejected not because the Framers of the Constitution doubted public intelligence but rather because they feared that without sufficient information about candidates from outside their State, people would naturally vote for a "favorite son" from their own State or region. At worst, no president would emerge with a popular majority sufficient to govern the whole country. At best, the choice of president would always be decided by the largest, most populous States with little regard for the smaller ones. [William C. Kimberling, *Essays in Elections: The Electoral College* (1992)]

Thus a system of "Electors" was chosen as a way to soften the influence of a raw-numbers-based demographic majority. This is the same reason America has a bicameral legislature with one chamber's members not apportioned by the population of the associated state.

Of course, the country (not to mention the Electoral College itself) has changed quite a bit since its founding. Mathematical gerrymandering. Multiple Voting Rights Acts. Expansive networks of Lobbyists and Political Action Committees. And, most prominently, a wider sense of who exactly is included in "The People" of the country. There have been so many transformations to what Americans see as their nation and its rules that it seems fair to ask whether the anti-popular spirit of the original Electoral system still applies.

In other words, there are excellent reasons to not be beholden to the political vision of people who lived two and a half centuries before you and who had contradictory and largely self-serving definitions of liberty and equality. However, it is also important to recognize that the Electoral College is serving its original purpose when it leads (albeit rarely) to an outcome counter to that of the popular vote.

2 Electoral Misalignment

OK so we know the Electoral College was never meant to align perfectly with the direction of the popular vote, but the two go in the same direction often enough that it is noteworthy when they don't. This leads to a basic question. Can we be more rigorous about how likely this lack of alignment will happen in a given election? Namely, for a given election (i.e., its polls, turnout stats, count of electors, etc.) can we determine the probability that whoever wins the Electoral College will also lose the popular vote?

To better outline this situation, we can represent the voting outcomes for a single candidate at the end of Election day.

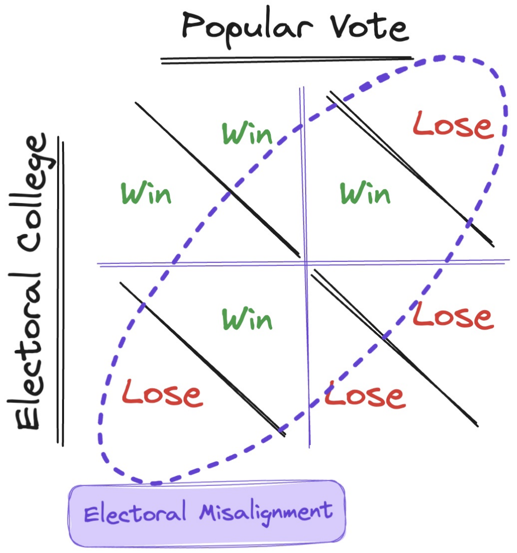


Figure 1: Election outcomes for a candidate. They can either win both the popular vote and the electoral college. Lose both, or win one and lose the other. When the candidate wins one and loses the other, we call the election "misaligned." We want to compute the probability of misalignment for the 2020 election.

Either the candidate wins both the popular vote and the Electoral College, the candidate loses both, or the candidate wins one and loses the other[1] . We call these "win-loss" situations "Electoral Misalignment" and we want to compute their total probability.

To do so, we will begin by outlining some naive but convenient theoretical assumptions of this calculation and will then focus on the 2020 election. We will pretend that we are just one day prior to that election day with all the then-current polling data. Here's what we want to know:

1. What was the misalignment probability in the 2020 election?
2. Was the misalignment probability higher for a Democratic presidential win or a Republican presidential win?
3. What does this suggest about the demographics of electoral politics in 2020 and potentially in the future?

3 The Model

There are undoubtedly many models of presidential elections, but we will start from scratch, laying out the assumptions and alternative routes (which we will note take) in building the final model.

We assume there are only two candidates running for president. A democratic candidate represented by D and a republican candidate represented by R. In actual elections, there are often third-party candidates but we will assume that we can split these candidates votes among the two candidates in ways that don't change the vote-differences for each state[2]. We define M as the number of "states" and label each state with $\alpha = 1, \ldots, M$. We put "states" in quotation marks to note that the M entities include not only states like Texas or California, but also the voting districts in Nebraska, Maine, and the District of Columbia. From henceforth, we will write states without quotations to represent this larger set.

The results of an election can be represented by $\mathbf{n}_D = (n_{D,1}, n_{D,2}, \ldots, n_{D,M})$ and $\mathbf{n}_R = (n_{R,1}, n_{R,2}, \ldots, n_{R,M})$ representing, respectively, the votes for the democratic and the republican candidate in each of the M states.

Our larger objective is to calculate the probability that a candidate will win the popular vote, the electoral vote, or generally any combination of winning one and losing the other. In order to do this, we need to associate probabilities with the particular votes for each state and define what it means to win the popular vote and the electoral vote.

3.1 Popular Win

In a popular vote system, the candidate who receives the most votes across all states wins. We denote by $n_D \equiv \sum_{\alpha=1}^{M} n_{D,\alpha}$ and $n_R \equiv \sum_{\alpha=1}^{M} n_{R,\alpha}$ the number of total votes for the Democratic and Republican candidates, respectively. Therefore we can write the condition for the Democratic candidate to win the popular vote as

[1]There is a small chance of tying in the Electoral College and an even smaller chance of tying in the popular vote, but we ignore these low likelihood events.

[2]This is admittedly a strong assumption (i.e., that an equal number of people would have voted for the democratic and republican candidates if the third parties were not available)

$$\sum_{\alpha=1}^{M}(n_{D,\alpha}-n_{R,\alpha})>0, \qquad [D \text{ Popular Vote Win}] \tag{1}$$

3.2 Electoral Win

In the Electoral College system, each state gets a certain number of "electors," and if one candidate wins a majority[3] of the vote in that state, then all of the electors vote for that candidate. The candidate with the most electoral votes across all states is deemed the winner of the Electoral College and the presidential election.

We define λ_α as the number of electoral votes for state α (e.g., $\lambda_{\text{Texas}}=38$). We denote e_D and e_R as the total number of electoral votes for the Democratic and Republican candidates respectively. Using the Heaviside step function $\Theta(x)$ defined as

$$\Theta(x)=\begin{cases}1 & \text{for } x>0\\ 0 & \text{otherwise}\end{cases} \tag{2}$$

we can write the number of electoral votes for the Democratic and Republican candidates as

$$e_D=\sum_{\alpha=1}^{M}\lambda_\alpha\Theta\left(n_{D,\alpha}-n_{R,\alpha}\right) \qquad e_R=\sum_{\alpha=1}^{M}\lambda_\alpha\Theta\left(n_{R,\alpha}-n_{D,\alpha}\right). \tag{3}$$

For a democratic candidate to win, we must have $e_D>e_R$, or

$$\sum_{\alpha=1}^{M}\lambda_\alpha H(n_{D,\alpha}-n_{R,\alpha})>0, \qquad [D \text{ Electoral College Win}] \tag{4}$$

where we defined $H(x)\equiv\Theta(x)-\Theta(-x)$.

3.3 Total Votes and Margin

Eq.(1) and Eq.(4) give us the conditions for a popular vote and Electoral College win in terms of the votes for Democratic and Republican candidates, but for the models we build later it will prove better to use a different set of variables. We define n_α as the total number of votes in a state and δ_α as the difference in the fraction of votes between the Democratic and Republican candidates:

$$n_\alpha\equiv n_{\alpha,D}+n_{\alpha,R} \qquad \delta_\alpha\equiv\frac{n_{\alpha,D}-n_{\alpha,R}}{n_{\alpha,D}+n_{\alpha,R}} \tag{5}$$

In polling speak, n_α is the raw "turnout" or "number of ballots" cast in an election, and δ_α is the margin of victory (or loss) for the Democratic candidate in state α. Inverting the system in Eq.(5) to solve for $n_{D,\alpha}$ and $n_{R,\alpha}$ in terms of n_α and δ_α, we find that the popular vote and Electoral College win conditions Eq.(1) and Eq.(4) become, respectively,

[3] Actually, it is if they win a "plurality" of the vote, but when we restrict the options to two candidates then this amounts to a majority.

$$\sum_{\alpha=1}^{M} n_\alpha \delta_\alpha > 0, \qquad \sum_{\alpha=1}^{M} \lambda_\alpha H(\delta_\alpha) > 0 \tag{6}$$

where we used the identity $H(n_\alpha \delta_\alpha) = H(\delta_\alpha)$ for $n_\alpha > 0$. Alternatively, using vector notation $\boldsymbol{\delta} = (\delta_1, \delta_2, \ldots, \delta_M)$, we can write simpler popular vote and Electoral College win conditions as

$$\mathbf{n} \cdot \boldsymbol{\delta} > 0, \qquad \boldsymbol{\lambda} \cdot H(\boldsymbol{\delta}) > 0, \tag{7}$$

where the $\Theta(\mathbf{x}) \equiv (\Theta(x_1), \Theta(x_2), \ldots, \Theta(x_M))$. Finally, using the Heaviside function Eq.(2), we can write these two conditions as binary yes or no (i.e., "1" or "0") functions:

$$\text{Dem Popular Win} = \Theta\left(\mathbf{n} \cdot \boldsymbol{\delta}\right) \tag{8}$$

$$\text{Dem Electoral College Win} = \Theta\left(\boldsymbol{\lambda} \cdot H(\boldsymbol{\delta})\right) \tag{9}$$

4 Probabilistic Models

The benefit in writing the conditions for a popular vote or Electoral College win in terms of $\boldsymbol{\delta}$ and $\mathbf{n}$ is that we can use polling data and previous voter turnout data to build probabilistic models for the election-day values of these quantities. Both win-margin and the total number of ballots in a state can be seen as random variables in that we do not have enough information to precisely specify them on the day of the election. But we can make best guesses as to their spaces of possible values using well-chosen assumptions and some historical data.

First, we define $\rho_0(\boldsymbol{\delta}, \mathbf{n})$ as the probability distribution for the margin and vote count vectors. With this probability density, we can use Eq.(8) to write the probability for a Democratic win of the popular vote. We have

$$\text{Prob}\left(n_D > n_R\right) = \int_{\Omega^M_{\text{margin}}} d^M\boldsymbol{\delta} \int_{\Omega^M_{\text{votes}}} d^M\mathbf{n}\, \rho_0(\boldsymbol{\delta}, \mathbf{n})\, \Theta\left(\boldsymbol{\delta} \cdot \mathbf{n}\right), \tag{10}$$

where $\Omega^M_{\text{diff}} = [-1, 1]^M$ and $\Omega^M_{\text{votes}} = \mathbb{R}^M_+$ are the domains of integration for the margin and total number of votes respectively.

Second, we will make three simplifying assumptions for the distribution ρ_0:

1. **Margin and Turnout Independence:** The random variables δ_α and n_α for the same state are independent (i.e., the margin of win and the number of ballots are independent)
2. **State Independence:** The random variables δ_α and $\delta_{\alpha'}$ for $\alpha \neq \alpha'$ are independent (i.e., different states have independent distributions)
3. **Normality:** Both random variables δ_α and n_α are normally distributed.

The first assumption allows us to factor the distribution between margin and vote counts:

$$\rho_0(\boldsymbol{\delta}, \mathbf{n}) = \rho_{\text{margin}}(\boldsymbol{\delta})\rho_{\text{votes}}(\mathbf{n}) \tag{11}$$

The last two assumptions then allow us to model the probability distributions for the margin and for the vote-counts across states as

$$\rho_{\text{margin}}(\boldsymbol{\delta}) \equiv \prod_{\alpha=1}^{M} \frac{1}{\sqrt{2\pi\sigma_\alpha^2}} e^{-(\delta_\alpha-\mu_\alpha)^2/2\sigma_\alpha^2}, \qquad \rho_{\text{votes}}(\mathbf{n}) \equiv \prod_{\alpha=1}^{M} \frac{1}{\sqrt{2\pi s_\alpha^2}} e^{-(n_\alpha-m_\alpha)^2/2s_\alpha^2}. \tag{12}$$

The parameters μ_α and σ_α are the expected value and the standard deviation of the Democratic candidate's margin of victory (or loss) for state α. And m_α and s_α are the expected value and the standard deviation for the total number of ballots cast in the state α for the two candidates.

With the probability distributions defined in Eq.(12) and the conditions of the popular vote or Electoral College win in Eq.(8) and Eq.(9), we can now write expressions for the probabilities of the Democratic candidate winning the popular vote or the Electoral College. For the popular vote, the win probability is[4]

$$\text{Prob}\,(n_D > n_R) = \int_{\mathbb{R}^{2M}} d^M\boldsymbol{\delta}\, d^M\mathbf{n}\, \rho_{\text{margin}}(\boldsymbol{\delta})\, \rho_{\text{votes}}(\mathbf{n})\, \Theta\,(\boldsymbol{\delta}\cdot\mathbf{n}) \qquad [D \text{ Electoral Win}] \tag{13}$$

and for the Electoral College, the win probability is

$$\text{Prob}\,(e_D > e_R) = \int_{\mathbb{R}^{2M}} d^M\boldsymbol{\delta}\, \rho_{\text{margin}}(\boldsymbol{\delta})\, \Theta(\boldsymbol{\lambda}\cdot H(\boldsymbol{\delta})) \qquad [D \text{ Popular Win}] \tag{14}$$

Now, the question that motivated this discussion went beyond the win-or-lose probability for each branch of the two voting systems. Instead, we wanted to know the probability that the voting systems were misaligned, namely that one candidate would win according to one and lose according to the other. We can again use our probability formalism to calculate this quantity. First, we note that the total misalignment probability consists of the sum of two terms:

$$\text{Misalignment Probability} = \text{Prob}\,(e_D > e_R \cap n_D < n_R) + \text{Prob}\,(e_D < e_R \cap n_D > n_R) \tag{15}$$

This is the probability that one candidate wins the electoral and loses the popular plus the probability that the other candidate gets the same outcome. In terms of our above probability distributions, the quantities that make up this expression can be written as

$$\text{Prob}\,(e_D > e_R \cap n_D < n_R) = \int_{\mathbb{R}^M} d^M\boldsymbol{\delta}\, d^M\mathbf{n}\, \rho_{\text{margin}}(\boldsymbol{\delta})\, \rho_{\text{votes}}(\mathbf{n})\, \Theta(\boldsymbol{\lambda}\cdot H(\boldsymbol{\delta}))\Theta\,(-\boldsymbol{\delta}\cdot\mathbf{n}) \tag{16}$$

$$\text{Prob}\,(e_D < e_R \cap n_D > n_R) = \int_{\mathbb{R}^M} d^M\boldsymbol{\delta}\, d^M\mathbf{n}\, \rho_{\text{margin}}(\boldsymbol{\delta})\, \rho_{\text{votes}}(\mathbf{n})\, \Theta(\boldsymbol{\lambda}\cdot H(-\boldsymbol{\delta}))\Theta\,(\boldsymbol{\delta}\cdot\mathbf{n})\,, \tag{17}$$

where we flipped the sign of $\boldsymbol{\delta}$ according to whether the R candidate winning corresponds to a "positive" voting margin. Beyond calculating misalignment probability, we can use this formalism to see whether the Electoral College supports or works against a candidate in terms of whether winning or losing the Popular vote is consistent with their win of the Presidential election.

If it is more likely for a candidate to win the Electoral College while losing the popular vote than it is for the candidate to lose the Electoral College while winning the popular vote, we can interpret this result as the Electoral College being biased (in the statistical sense5) towards that

[4]In the integrals, we are integrating over all of $\mathbb{R}^M$ space but we know the margin is bounded above by 1 and below by 0 while the number of votes is bounded below by 0. We assume that the widths of the distributions are sufficiently small that these boundaries are essentially at infinity.

candidate. That is, the Electoral College is more likely to not reflect the will of the people when said candidate is elected than when the competing candidate is elected.

We can measure the extent of the bias by computing the normalized difference between each term that makes up the misalignment probability. We define this bias as

$$D - R \text{ Electoral College Bias} = \frac{\text{Prob}\,(e_D > e_R \cap n_D < n_R) - \text{Prob}\,(e_D < e_R \cap n_D > n_R)}{\text{Prob}\,(e_D > e_R \cap n_D < n_R) + \text{Prob}\,(e_D < e_R \cap n_D > n_R)} \tag{18}$$

If $D - R$ Electoral College Bias $= 1$ then the Electoral College was completely biased (again in the statistical sense[5]) towards the Democratic candidate. This means that there was no way for the Republican candidate to win the Electoral College without also winning the popular vote, but the Democratic Candidate could win the Electoral College without also winning the popular vote. A value of -1 corresponds to the opposite situation, and a value of 0 means there is no bias, i.e., it's equally likely for either candidate to win the Electoral College while losing the popular vote.

With these definitions, we can now start collecting data to estimate misalignment probabilities and biases. We will use the 2020 election as our frame of reference.

5 Data Collection and Parameter Estimates

Let's imagine it is November 2nd, 2020, the night before the Presidential election. We want to use the above model to predict not only the probability that one candidate will win the Electoral College but also the probability that the Election results will be "misaligned," i.e., that the candidate who wins the Electoral College also loses the popular vote. We want the total probability this will occur for either candidate and beyond this, we want to know for which candidate such misalignment is more likely. Knowing the latter will let us know whether this election's electoral map has an anti-popular bias for one candidate or the other.

From Eq.(9), it is clear that we will need to determine the quantities μ_α, σ_α, m_α, and s_α for all states α. That is, we want the expected values and widths of the margin and total number of votes for each state. The two data sources we will use to estimate these quantities are the 2020 polling results from 270towin.com and vote count statistics for the 2000 to 2016 Presidential elections (remember we're pretending we don't know the votes for 2020).

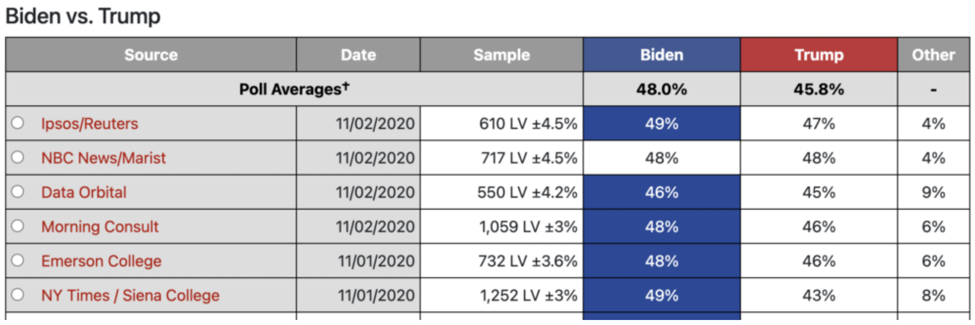
5.1 Margin Data Collection

We want to estimate the mean and variance (i.e., μ_α and σ_α^2) of

$$\delta_\alpha \equiv \frac{n_{\alpha,D} - n_{\alpha,R}}{n_{\alpha,D} + n_{\alpha,R}} \tag{19}$$

for all states α. To do so, we can collect pre-election polling data for each state, compute the difference between the Democratic and Republican vote percentages, and then compute the relevant statistics for the computed differences. For example, on the site 270towin.com/2020-polls-biden-trump/arizona/ we see the following Arizona pre-election polling table

[5]In statistics, bias means that the computed statistic (in this case the electoral college vote) is not an accurate reflection of the population. https://en.wikipedia.org/wiki/Bias_(statistics)



Biden vs. Trump

Source	Date	Sample	Biden	Trump	Other
Poll Averages†			48.0%	45.8%	-
Ipsos/Reuters	11/02/2020	610 LV ±4.5%	49%	47%	4%
NBC News/Marist	11/02/2020	717 LV ±4.5%	48%	48%	4%
Data Orbital	11/02/2020	550 LV ±4.2%	46%	45%	9%
Morning Consult	11/02/2020	1,059 LV ±3%	48%	46%	6%
Emerson College	11/01/2020	732 LV ±3.6%	48%	46%	6%
NY Times / Siena College	11/01/2020	1,252 LV ±3%	49%	43%	8%

Figure 2: Pre-election day 2020 polling for Arizona. We can compute the difference in percentages between Biden and Trump and use the results from the polls as samples drawn from the true election-day distribution

From this table, it is straightforward to compute the Biden-Trump margin for each poll and then compute the mean and variance across the five most recent polls relative to the election day. These quantities will then serve as estimates μ_{Arizona} and $\sigma^2_{\text{Arizona}}$. To compute the full set of μ_α and σ_α, we just need to do this for all states.

To streamline this process, we can write a script to scrape the 270towin site and then store the computed quantities in a dictionary. Here is an example section of the script

```python
# whether to include conservative correction
correction_ = True
size_ = 0.03

# going through state list for non-congressional districts
# compiling mean and median data
for state in tqdm(state_list):
    # eliminates the congressional districts
    if sum([state.find('1'), state.find('2'), state.find('3')])==-3:
        # get response for website
        state_short = reduced_state_dict[state]
        wikiurl=f"https://www.270towin.com/2020-polls-biden-trump/{state_short}/"
        response=requests.get(wikiurl)

        # parse data from the html into a beautifulsoup object
        soup = BeautifulSoup(response.text, 'html.parser')
        find_table=soup.find_all('table',{'id':"polls"})

        # getting first table
        df=pd.read_html(str(find_table[0]))
        # convert list to dataframe
        df=pd.DataFrame(df[0])

        # computing biden trump poll difference
        df['Diff'] = (df['Biden'].str.strip('%')
        .astype(float)-df['Trump'].str.strip('%').astype(float)[0])/100

        # removing the header and getting first five polls
        # offsetting index if 'averages' is first elemenat
        idx0 = sum([True for elem in list(df['Source']) if 'verage' in str(elem)])
        df_cut = df.iloc[idx0:idx0+5]

        # computing mean and standard deviation of most recent five polls
        mean_ = np.mean(df_cut['Diff'])
        var_ = np.var(df_cut['Diff'])

        # filling in delta dictionary
        # incorporating hidden conservative lean
        delta_dict[state]['mean'] = conserv_correc(mean_, correction_, size_)
        delta_dict[state]['var'] = var_
```

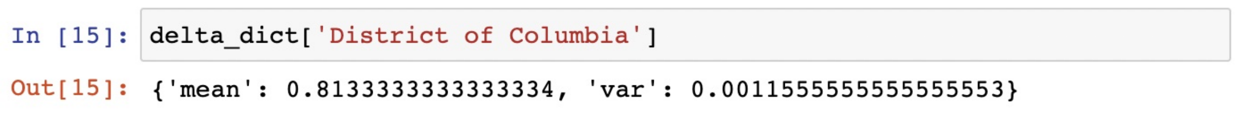
(***Note:*** *This code is part of a larger notebook and will not run on its own. See the notebook at the end for a full executable file.*)

One thing that should be mentioned about the above code is the function `conserv_correc`. The function is defined as

```python
def conserv_correc(mean, include=True, size=0.03):
    """
    Polls today (i.e., circa 2020) seem to
    underestimate conservative preference. We
    include a small correction to account for this bias
    """
    if include:
        return mean - size
    else:
        return mean
```

The function shifts the average margin of the polls by a certain amount to account for the fact that the 2016 polls underestimated the lean toward Republican candidates. We assume that the 2020 polls, and we account for this underestimation by this small shift.

With the mean and variance of the polling margins for various states, we can now access a single state to determine its relevant statistics. For example, for Washington D.C. we have

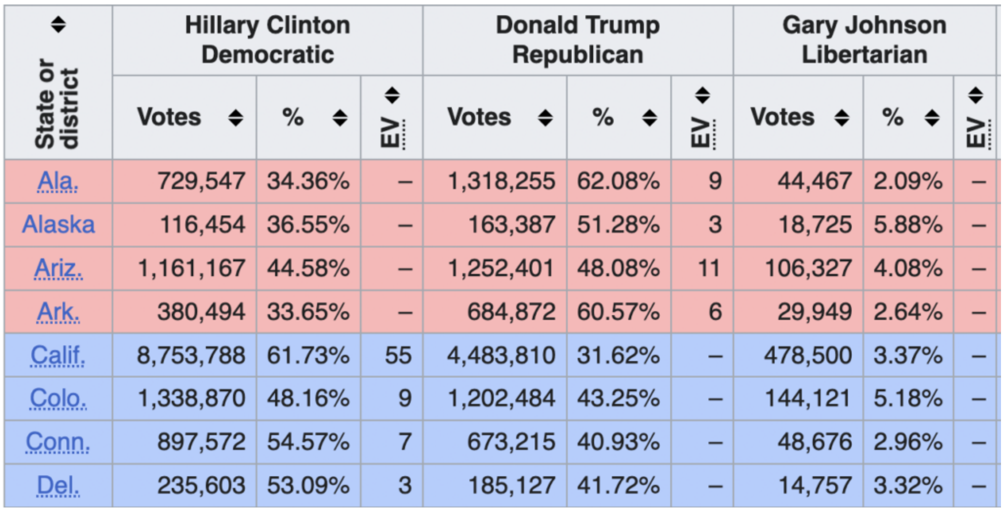


```
In [15]: delta_dict['District of Columbia']

Out[15]: {'mean': 0.8133333333333334, 'var': 0.0011555555555555553}
```

5.2 Total-Votes Data Collection

For the total-votes data collection, we want to estimate the mean m_α and variance s_α^2 of the total number of votes for each state in the 2020 election. I wasn't able to find data on pre-election day turnout projections for 2020, so we will take a more predictive approach: For each state, we will collect the total number of ballots cast in the presidential elections from 2000 to 2016, and we will use those five data points to train a linear regression that forecasts the turnout in 2020. The 2020 prediction for state α will stand in for the mean m_α and the mean squared error of the model will stand in for the variance s_α^2.

To collect this data, we again perform some web scraping, but we will use Wikipedia as our data source. For the 2000 to 2016 election years, Wikipedia keeps track of the number of ballots cast in each state. For example, a section of the 2016 table looks like



State or district	Hillary Clinton Democratic			Donald Trump Republican			Gary Johnson Libertarian		
	Votes	%	EV	Votes	%	EV	Votes	%	EV
Ala.	729,547	34.36%	–	1,318,255	62.08%	9	44,467	2.09%	–
Alaska	116,454	36.55%	–	163,387	51.28%	3	18,725	5.88%	–
Ariz.	1,161,167	44.58%	–	1,252,401	48.08%	11	106,327	4.08%	–
Ark.	380,494	33.65%	–	684,872	60.57%	6	29,949	2.64%	–
Calif.	8,753,788	61.73%	55	4,483,810	31.62%	–	478,500	3.37%	–
Colo.	1,338,870	48.16%	9	1,202,484	43.25%	–	144,121	5.18%	–
Conn.	897,572	54.57%	7	673,215	40.93%	–	48,676	2.96%	–
Del.	235,603	53.09%	3	185,127	41.72%	–	14,757	3.32%	–

Figure 3: Screenshot of Wiki table of 2016 US Election votes for major candidates. By assembling the total votes across states and for multiple election years, we can predict the turnout in 2020. (The total number of votes is at the right end of the table and not shown in the image)

By scraping this table for each year, we can compute the total number of votes for each state for each presidential election. The code to do this looks like

```python
# list of states in alphabetical order
state_list = list(electoral_votes.keys())

# years in string and integer form
years_string = ['2000', '2004', '2008', '2012', '2016']
years_int = np.array([int(year) for year in years_string])

# votecount dictionary of dataframes
votecount_df_dict = dict()

for year in tqdm(years_string):
    wikiurl
    =f"https://en.wikipedia.org/wiki/{year}_United_States_presidential_election"
        + "Results_by_state"
    response=requests.get(wikiurl)

    # parse data from the html into a beautifulsoup object
    soup = BeautifulSoup(response.text, 'html.parser')
    find_table=soup.find_all('table',{'class':"wikitable"})

    # getting table with electoral votes
    for table in find_table:
        if 'Iowa' in str(table) and 'Alabama' in str(table):
            table_key = table

    # getting first table
    df=pd.read_html(str(table_key))
    # convert list to dataframe
    df_orig=pd.DataFrame(df[0])

    # dropping the higest level column
    df_drop = df_orig.copy()
    df_drop.columns = df_orig.columns.droplevel()

    # converting state/district name to just state
    df_drop.rename(columns = {df_drop.columns[0]: 'State'}, inplace = True)

    # getting starting index for state names
    row_names = list(df_drop['State'])
    for k in range(len(row_names)):
        if 'Ala' in str(row_names[k]):
            start_idx = k
            break

    # getting column name for vote count
    first_level_names = list(df_orig.columns.droplevel(1))
    second_level_names = list(df_orig.columns.droplevel(0))
    for elem1, elem2 in zip(first_level_names, second_level_names):
        if 'Total' in elem1:
            break

    # getting compiling dictionary
    data_dict = {'State': state_list,
    'Total Votes': np.array(list(df_orig.iloc[k:k+56][(elem1, elem2)])).astype(int)}

    # creating dataframe for year
    votecount_df_dict[year] = pd.DataFrame.from_dict(data = data_dict)
```

(***Note:*** *This code is part of a larger notebook and will not run on its own. See the notebook at the end for a full executable file.*)

The total vote count data for each state in a particular `year` is stored in `votecount_df_dict[year]`. With this data, we can then predict the total number of votes for 2020 and the variance of the prediction. To visually depict this projection, we can plot, for example, the 2020 total votes projection for Wyoming compared with the actual value and the values from previous years.

In the case of Fig. 4, we see that the true 2020 value exists outside the 68% confidence interval for the prediction. Still, we will use this simple extrapolation model to predict turnout for each state.

With the mean and variance of the total votes for various states, we can now access a single state to determine its relevant statistics. For Wyoming, for example, we have

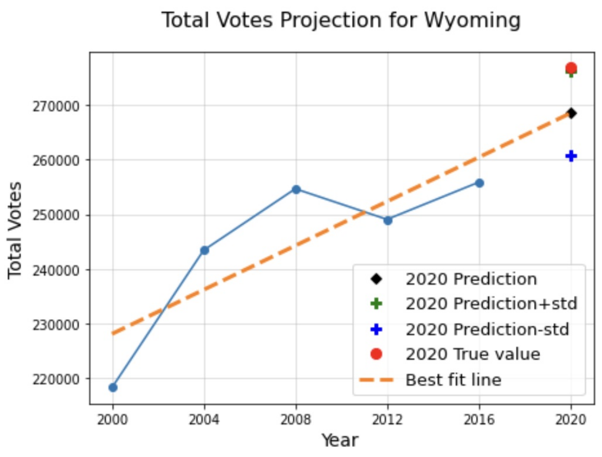


Figure 4: Projection of Wyoming's votes in 2020 using election data from 2000-2016. We can use such projections across all states to estimate the number of votes in 2020.

```
In [6]: # count of votes; Testing Wyoming
        votecount_dict['Wyoming']

Out[6]: {'mean': 268458.1000000001, 'var': 57467485.02}
```

6 Simulation

Having estimated μ_α, σ_α^2, m_α, and s_α^2 for each state α, we now know the probability distributions in Eq.(12), and we can compute Eq.(8) and Eq.(9), the probabilities of a popular vote and Electoral College win, respectively. Since both quantities are integrations over probability distributions, we can use Monte Carlo integration to evaluate them. Namely, rather than computing the integrals through a standard numerical quadrature, we can sample the space of points according to the probability distribution, compute the non-distribution integrand for each sample, and take the average of the result.

First, to define the parameters μ_α, σ_α^2, m_α, and s_α^2 from the collected data we convert the collected data into vectors and matrices

```python
# ballot count dictionary; assuming independent variances
n_mean_vec = np.array([votecount_dict[state_]['mean'] for state_ in state_list])
n_cov_matrix = np.diag([votecount_dict[state_]['var'] for state_ in state_list])

# delta dictionary; assuming independent variances
delta_mean_vec = np.array([delta_dict[state_]['mean'] for state_ in state_list])
delta_cov_matrix =  np.diag([delta_dict[state_]['var'] for state_ in state_list])

# electoral college vector
lambda_vector = np.array([electoral_votes[state_] for state_ in state_list])
```

Then we can simulate the possibilities, with the following code.

```python
##
# Democrat Win Calculation
##


# defining H function
H = lambda x: np.heaviside(x, 0)-np.heaviside(-x, 0)

# for sampling from normal distribution
sample_vector = lambda mean, cov: np.random.multivariate_normal(mean = mean, cov =
↪  cov)

# number of times to simulate election
Nsim = 10000

# lambda vector
lambda_vec = list()

# winning electoral college
dems_electoral_wins = list()

# winning popular vote
dems_popular_wins = list()

# winning both electoral college and popular vote
dems_elec_and_pop_wins = list()

# differences
pop_vote_diff = list()
elec_vote_diff = list()

# going through simulations
for _ in tqdm(range(Nsim)):

    # ballot count vector
    n_vector = sample_vector(n_mean_vec, n_cov_matrix)

    # difference vector
    delta_vector = sample_vector(delta_mean_vec, delta_cov_matrix)

    # popular vote win
    popular_win = np.heaviside(np.dot(delta_vector, n_vector), 0)

    # electoral vote win
    electoral_win = np.heaviside(np.dot(lambda_vector, H(delta_vector)), 0)

    # electoral and popular vote win
    electoral_popular_win = popular_win*electoral_win

    # appending election statistics
    pop_vote_diff.append(np.dot(delta_vector, n_vector))
    elec_vote_diff.append(np.dot(lambda_vector, H(delta_vector)))

    # appending election result
    dems_electoral_wins.append(electoral_win)
    dems_popular_wins.append(popular_win)
    dems_elec_and_pop_wins.append(electoral_popular_win)
```

From these simulations, we can compute the probability that the 2020 Democratic candidate (Joe Biden) had for winning either the Popular vote or the Electoral College.

```
In [19]: # printing probabilities
         print(f'Democrat Probabilities')
         print(f'***************')
         print(f'Probability of Electoral College Win:
         {round(np.sum(dems_electoral_wins)/Nsim, 4)}')
         print(f'Probability of Popular Vote Win:
         {round(np.sum(dems_popular_wins)/Nsim, 4)}')
```

```
Democrat Probabilities
***************
Probability of Electoral College Win: 0.7811
Probability of Popular Vote Win: 1.0
```

This model suggests that given pre-election Polling data (including the "conservative correction") Biden had an 78% chance of winning the Electoral College, but was essentially guaranteed to win the Popular Vote. Another way to see this result is to show histogram distributions of the Popular and Electoral College vote differences.

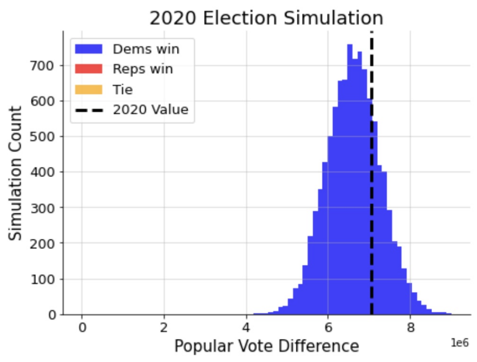


Figure 5

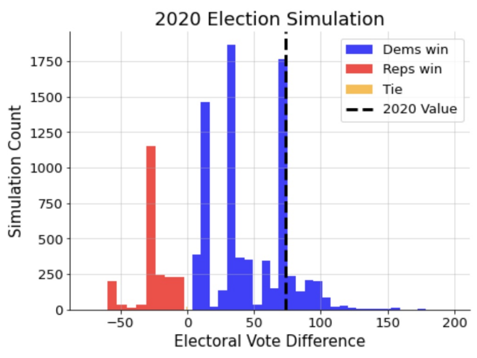


Figure 6

We see that the Democrat-Republican difference in popular votes is normally distributed with a peak at 7 million votes[6] with no part of the distribution extending below zero. On the other hand, the Democract-Republican difference in electoral votes, although showing the Democratic candidate winning most of the time, shows a non-negliglible number of of instances of a Republican win.

[6]The actual popular vote difference was $81,283,501 - 74,223,975 = 7,059,526$ [Source Link]

This result already hints at the answer to a question that started this investigation. If it was possible for the Republican candidate to have won the electoral college vote but almost impossible for said candidate to have also won the popular vote, then there must be a non-zero probability of misalignment in this election and moreover, the total misalignment probability must have come primarily from only one candidate's misaligned win.

7 Non-Symmetric Misalignment

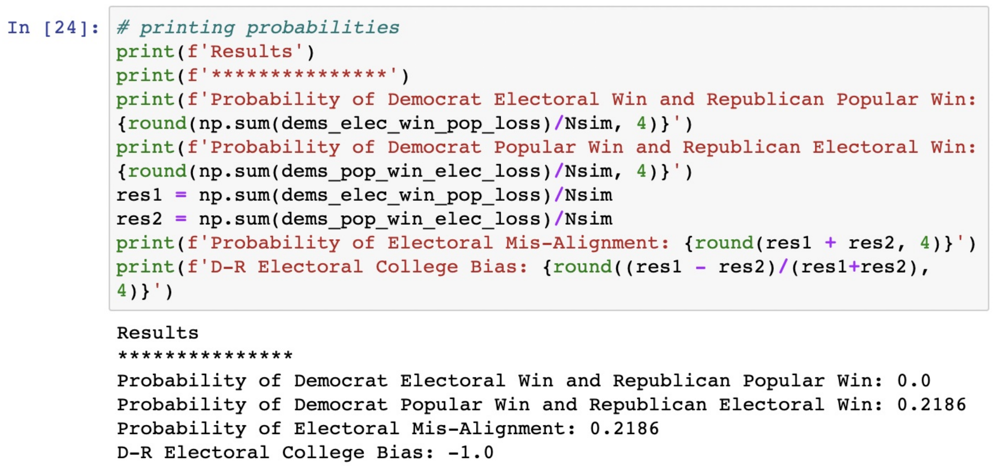
At last, we can turn to the question that prompted this investigation. We rewrite the question in the context of the 2020 election for convenience.

What was the probability of electoral misalignment for the 2020 election and which candidate was more likely to be misaligned if they won the electoral vote?

To answer this question, we can use a similar simulation script as that given above, but tailor it to calculate the joint probabilities Eq.(16) and Eq.(17):

```
##
# Misalignment Probability Calculation
##

# defining H function
H = lambda x: np.heaviside(x, 0)-np.heaviside(-x, 0)

# number of times to simulate election
Nsim = 10000

# lambda vector
lambda_vec = list()

# winning electoral college
dems_electoral_wins = list()

# winning popular vote
dems_popular_wins = list()

# winning electoral college and not winning popular vote
dems_elec_win_pop_loss = list()

# winning popular vote and not winning electoral vote
dems_pop_win_elec_loss = list()

# going through simulations
for _ in tqdm(range(Nsim)):

    # ballot count vector
    n_vector = sample_vector(n_mean_vec, n_cov_matrix)

    # difference vector
    delta_vector = sample_vector(delta_mean_vec, delta_cov_matrix)

    # popular vote
    dems_popular_win = np.heaviside(np.dot(delta_vector,  n_vector), 0)
    reps_popular_win = np.heaviside(np.dot(-delta_vector, n_vector), 0)

    # electoral vote
    dems_electoral_win = np.heaviside(np.dot(lambda_vector, H(delta_vector)), 0)
    reps_electoral_win = np.heaviside(np.dot(lambda_vector, H(-delta_vector)), 0)

    # appending election results
    dems_elec_win_pop_loss.append(dems_electoral_win*reps_popular_win)
    dems_pop_win_elec_loss.append(reps_electoral_win*dems_popular_win)
```

From running these simulations, we find



```
In [24]: # printing probabilities
         print(f'Results')
         print(f'***************')
         print(f'Probability of Democrat Electoral Win and Republican Popular Win:
         {round(np.sum(dems_elec_win_pop_loss)/Nsim, 4)}')
         print(f'Probability of Democrat Popular Win and Republican Electoral Win:
         {round(np.sum(dems_pop_win_elec_loss)/Nsim, 4)}')
         res1 = np.sum(dems_elec_win_pop_loss)/Nsim
         res2 = np.sum(dems_pop_win_elec_loss)/Nsim
         print(f'Probability of Electoral Mis-Alignment: {round(res1 + res2, 4)}')
         print(f'D-R Electoral College Bias: {round((res1 - res2)/(res1+res2),
         4)}')

         Results
         ***************
         Probability of Democrat Electoral Win and Republican Popular Win: 0.0
         Probability of Democrat Popular Win and Republican Electoral Win: 0.2186
         Probability of Electoral Mis-Alignment: 0.2186
         D-R Electoral College Bias: -1.0
```

These results show us two things: First, the misalignment probability was about 22% for the 2020 election; Second, the entirety of this misalignment came from the Republican candidate winning the Electoral College but losing the Popular vote (which is equivalent to the Democratic candidate winning the Popular Vote but Lossing the Electoral College). The fact that the $D-R$ Electoral College Bias is equal to -1 is another way to represent this fact. It implies that in all the cases where the Electoral College went against the Popular vote, the Republican candidate won the presidency and the Democratic candidate lost.

In other words, the composition of the likely Electoral College votes had an anti-popular bias against the Democratic candidate and in favor of the Republican candidate. The Republican candidate could win the election without winning the popular vote, but this was not true for the Democratic candidate. We can see this more clearly by redrawing our square diagram with some numbers filled into the boxes in Fig. 7.

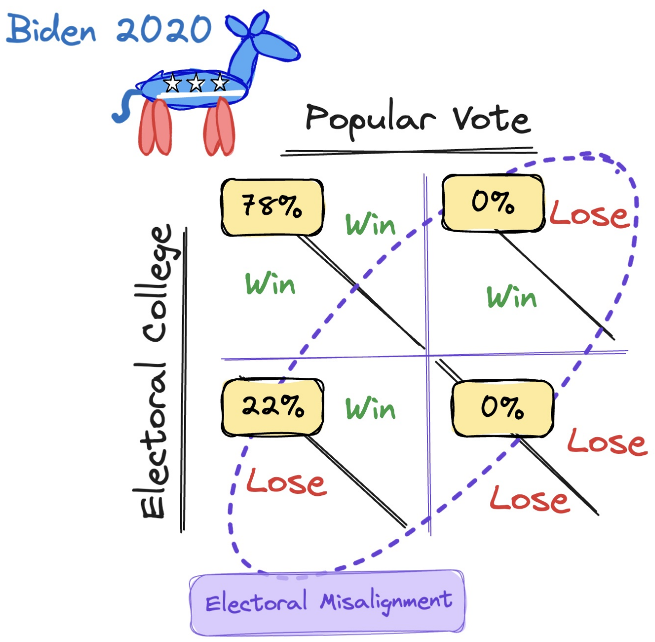


Figure 7: Election outcomes and probabilities for Biden 2020. Biden was essentially guaranteed to win the popular vote, but there was a possibility he would lose the Electoral College. This was not the case for Trump.

The model suggests that Biden was essentially guaranteed to win the popular vote, but not guaranteed (albeit a likely favorite) to win the Electoral College.

When one reflects on the origins of the Electoral College system and on the nature of the 2020 (or 2016) election, one sees an irony in the lopsided nature of this misalignment. The Electoral College system was instituted largely to avoid electing into positions of power the sort of anti-elite populist candidate that Donald Trump's candidacy represented (albeit superficially).

Describing the origins of this system, Phillip J VanFossen notes that the Constitutional founders

> ...believed that the electors would ensure that only a qualified person became president. And they thought the Electoral College would serve as a check on a public who might be easily misled. [VanFossen, Phillip J. (November 4, 2020). "Who invented the Electoral College?". *The Conversation*.]

Thus the Electoral College, by definition, was supposed to be "election according to an established elite." And yet in 2016 and 2020, more than 200 years from that time, it is that very system that made it more likely than otherwise that a populist candidate would be elected to the Presidency against the majority-count desires of the wider population.