

## Imports

```
In [1]: import numpy as np
import scipy as sp
import scipy.optimize
import matplotlib.pyplot as plt
```

## Import data

<https://www.cset-foretell.com/questions/74-what-will-the-combined-revenue-of-alphabet-amazon-apple-facebook-and-microsoft-be-in-the-first-two-quarters-january-1-through-june-30-of-2021> (<https://www.cset-foretell.com/questions/74-what-will-the-combined-revenue-of-alphabet-amazon-apple-facebook-and-microsoft-be-in-the-first-two-quarters-january-1-through-june-30-of-2021>)

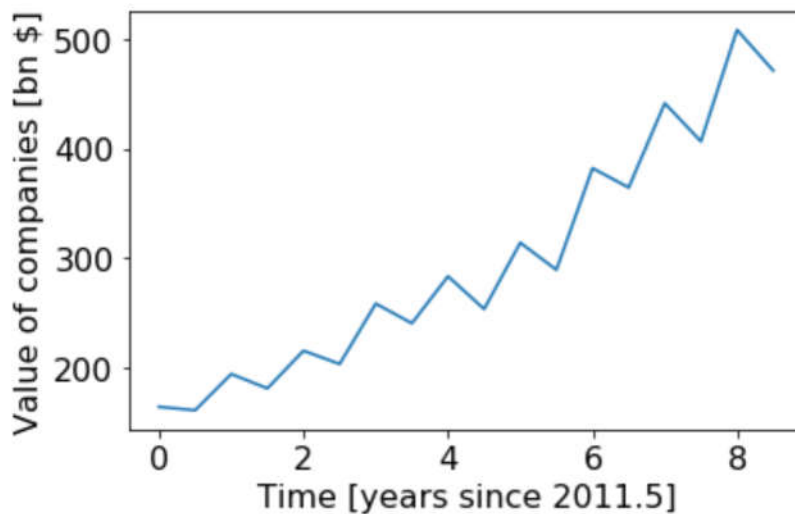
```
In [104]: # Data from https://docs.google.com/spreadsheets/d/1a_YA8iTG3IBMzTScxE
7LbP-8c2vYT336L5aVqMYq3Mc/edit#gid=526298235
data = {"date": np.array([2011.5, 2012.0, 2012.5, 2013.0, 2013.5, 201
4.0, 2014.5, 2015.0, 2015.5, 2016.0, 2016.5,
2017.0, 2017.5, 2018.0, 2018.5, 2019.0, 201
9.5, 2020.0]) - 2011.5,
        "quant": np.array([163.56, 160.39, 193.59, 180.41, 214.86, 202.
73, 257.98, 240.00, 283.00, 253.00, 313.75,
288.90, 381.92, 364.23, 441.38, 406.46, 508.
55, 471.30])}

xlabel = 'Time [years since 2011.5]'
ylabel = 'Value of companies [bn $]'
```

```
In [105]: plt.rcParams.update({'font.size': 16})
fig, ax = plt.subplots(1,1)

ax.plot(data['date'], data['quant'])
ax.set_xlabel(xlabel)
ax.set_ylabel(ylabel)

plt.tight_layout()
```



## Generate trend-fitting, exponential, H1 only

```
In [106]: def outside_func(t, a, tau, b):
            return a * np.exp(t/tau) + b
```

```
In [107]: a_guess = 100
tau_guess = 5
b_guess = 100
popt, pcov = sp.optimize.curve_fit(outside_func, data['date'][1::2], d
ata['quant'][1::2], p0=[a_guess, tau_guess, b_guess])
perr = np.sqrt(np.diag(pcov)) # The standard deviations of the paramet
ers
```

```

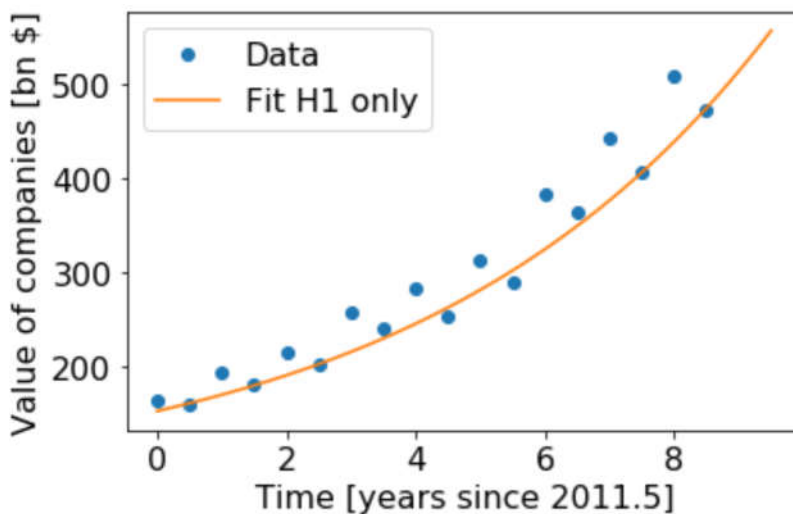
In [109]: fig, ax = plt.subplots(1,1)

t_final = 9.5
fit_label = 'Fit H1 only'
ax.plot(data['date'], data['quant'], 'o', label='Data')
t_plot = np.linspace(0, t_final, 100)
ax.plot(t_plot, outside_func(t_plot, *popt), label=fit_label)
ax.set_xlabel(xlabel)
ax.set_ylabel(ylabel)

ax.legend()

plt.tight_layout()

```



```

In [110]: popt

```

```

Out[110]: array([85.30184976,  5.44455158, 67.66551325])

```

```

In [111]: pcov

```

```

Out[111]: array([[ 8.44950730e+02,  2.67595338e+01, -1.04135961e+03],
                  [ 2.67595338e+01,  8.55533343e-01, -3.26442072e+01],
                  [-1.04135961e+03, -3.26442072e+01,  1.30850940e+03]])

```

## Monte Carlo of trends

I think with the fit we could probably just build up the probability density function  $P(y) = \Pr(f(x) = y)$  for the ranges of  $y$  we care about, just given our fit parameters and uncertainty for  $x$ . However, I think an equivalent way to do this is to just run a monte carlo simulation were we randomly sample from our fit parameters, and then bin the value of  $f(x)$  that we get from the simulation.

To do this properly, we need to really consider the full covariance matrix of the fit parameters. However, I think Wikipedia just gives the formula for the joint distribution of the fit parameters given the covariance matrix here: [https://en.wikipedia.org/wiki/Covariance\\_matrix#Covariance\\_matrix\\_as\\_a\\_parameter\\_of\\_a\\_distribution](https://en.wikipedia.org/wiki/Covariance_matrix#Covariance_matrix_as_a_parameter_of_a_distribution) ([https://en.wikipedia.org/wiki/Covariance\\_matrix#Covariance\\_matrix\\_as\\_a\\_parameter\\_of\\_a\\_distribution](https://en.wikipedia.org/wiki/Covariance_matrix#Covariance_matrix_as_a_parameter_of_a_distribution)). However, I think that is just the formula for the multivariate normal distribution, which numpy already has a built in function to sample from

```
In [112]: # def gen_params_dist(x_arr, popt, pcov):  
#         """ Find the probability of a set of parameters being the  
#             best fit to a dataset given the best fit parameters and  
#             covariance matrix """  
#         n = len(popt)  
#         sig_det = np.linalg.det(pcov)  
#         sig_inv = np.linalg.inv(pcov)  
  
#         pre_factor = (2*np.pi)**(-n / 2) * sig_det**(-0.5)  
#         exp_part = np.dot(sig_inv, (x_arr - popt))  
#         exponent = -0.5 * np.dot((x_arr - popt), exp_part)  
#         return pre_factor * np.exp(exponent)
```

```
In [114]: # Generate fits

n_fits = 1000
# For now, assume that the fit parameters will be gaussian distributed
# about the value in popt with the standard deviation the value of perr
# fit_params = np.array([np.random.normal(opt, err, n_fits) for opt, e
rr in zip(popt, perr)])
fit_params_gen = np.random.multivariate_normal(popt, pcov, n_fits)

# Clean to fit based on some common sense criteria. Be very careful th
rowing only a lot of data though!!
fit_params = []
n_removed = 0
for fit_list in fit_params_gen:
    if fit_list[0] < 0:
        n_removed += 1
        n_fits -= 1
    else:
        fit_params.append(fit_list)

print('Generated %i plots, discarded %i' %(n_fits + n_removed, n_remov
ed))
# Plot MC simulations and build MC data
fig, ax_mc = plt.subplots(1,1)

final_vals = []
for i, popt_mc in enumerate(fit_params):
    label = 'MC simulation' if i == 0 else None
    t_plot = np.linspace(0, t_final, 100)
    ax_mc.plot(t_plot, outside_func(t_plot, *popt_mc), 'k', alpha=0.1,
label=label)

    final_vals.append(outside_func(np.array([t_final]), *popt_mc)[0])

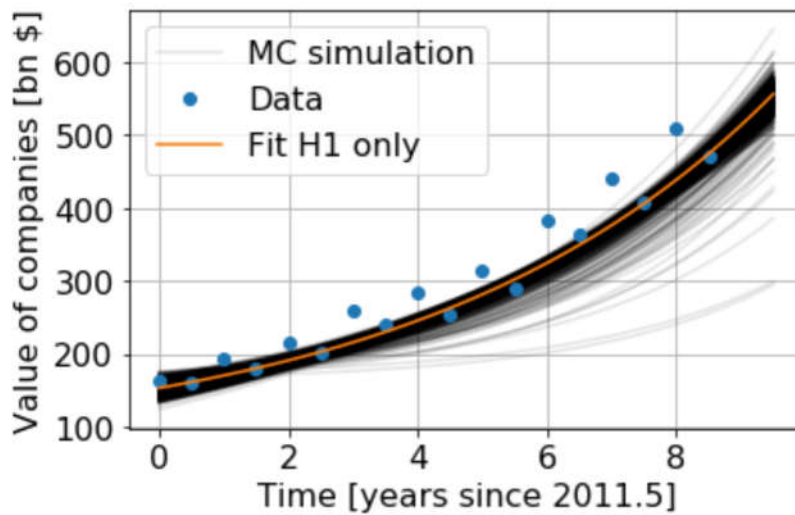
# Plot original data and fit
ax_mc.plot(data['date'], data['quant'], 'o', label='Data')
t_plot = np.linspace(0, t_final, 100)
ax_mc.plot(t_plot, outside_func(t_plot, *popt), label=fit_label)

ax_mc.set_xlabel(xlabel)
ax_mc.set_ylabel(ylabel)

ax_mc.grid()
ax_mc.legend()

plt.tight_layout()
```

Generated 1000 plots, discarded 1



**Including histogram**

```

In [126]: # Plot MC simulations and build MC data
cset_boundaries = [440, 490, 540, 590]

fig, [ax_mc, ax_hist] = plt.subplots(1, 2, figsize=(8, 4), gridspec_kw=
    ={'width_ratios': [3, 1]}, sharey=True)

for i, popt_mc in enumerate(fit_params):
    label = 'MC simulation' if i == 0 else None
    t_plot = np.linspace(0, t_final, 100)
    ax_mc.plot(t_plot, outside_func(t_plot, *popt_mc), 'k', alpha=0.1,
        label=label)

# Plot original data and fit
ax_mc.plot(data['date'], data['quant'], 'o', label='Data')
t_plot = np.linspace(0, t_final, 100)
ax_mc.plot(t_plot, outside_func(t_plot, *popt), label=fit_label)

# for boundary in cset_boundaries:
#     ax_mc.axhline(boundary, color='r', linewidth=0.3)

ax_mc.set_xlabel(xlabel)
ax_mc.set_ylabel(ylabel)
# ax_mc.set_yscale('log')

ax_mc.grid()
ax_mc.legend()

# Plot histogram
bin_min = min(final_vals)
bin_max = max(final_vals)

bins = np.concatenate([[bin_min], cset_boundaries, [bin_max]])

n, bins, patches = ax_hist.hist(np.array(final_vals), bins, color='k',
    orientation='horizontal')
# ax_hist.set_ylim(ax_mc.get_ylim())

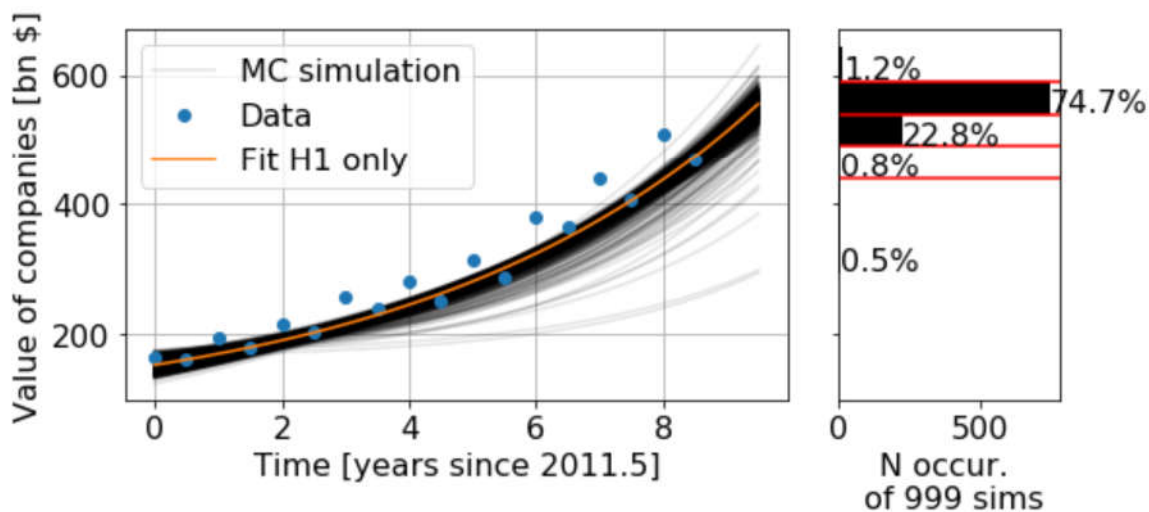
ax_hist.set_xlabel('N occur. \n of {} sims'.format(n_fits))

for boundary in cset_boundaries:
    ax_hist.axhline(boundary, color='r')

text_offset = (max(bins) - min(bins)) / 100
percentages = [n_val / n_fits * 100 for n_val in n]
for n_val, bin_min, percent in zip(n, bins[:-1], percentages):
    ax_hist.text(n_val, bin_min + text_offset, '%.1f%%' % (percent))

plt.tight_layout()

```



This seems like there is a trend for the poorly fit lines to be below the trend line...

## Generate final forecast

Before I saw any data, I would have put equal confidence in each of the 5 bins, so my prior was {0.2, 0.2, 0.2, 0.2, 0.2}.

I then tried to incorporate information about historical trends alone by doing a linear fit on the data, and estimating the distribution for 2021-H1 given the uncertainty in this exponential fit. From this curve fitting, my best estimate of the probabilities of the trend reaching the values {<440bn, 440-490bn, 490-540bn, 540-590bn, >590bn} is {0.5%, 0.8%, 22.8%, 74.7%, 1.2%} respectively (For how I arrived at these, see <https://github.com/cphenicie/forecasts/tree/main/Foretell> (<https://github.com/cphenicie/forecasts/tree/main/Foretell>) in the file 2020923\_value-of-tech-companies).

I arbitrarily assigned 50% confidence to my prior values and 50% confidence to these trend-fits values. (I think in most years one could be more than 50% confidence in an exponential extrapolation of a company's valuation, but with COVID I'll assume more ignorance.) I then arrived at updated values by taking the mean of the prior with the trend-fit value ( $0.5 [prior] + (1-0.5) [trend-fit \text{ value}]$ ), and then normalizing the resulting distribution

Finally, I'm unsure how to incorporate any other information into this, I could see arguments for even greater growth (with the promise of a vaccine) or less growth (with the reality of the pandemic still being an issue during 2021-H1)

```
In [125]: prior = [0.2, 0.2, 0.2, 0.2, 0.2]
          fit = [0.005, 0.008, 0.228, 0.747, 0.012]
          c_fit = 0.5
          mean_vals = [(1-c_fit) * p + c_fit * f for p, f in zip(prior, fit)]
          final_dist = mean_vals / np.sum(mean_vals)
          final_dist
```

```
Out[125]: array([0.1025, 0.104 , 0.214 , 0.4735, 0.106 ])
```

```
In [ ]:
```