Lab08

November 8, 2017

```
In [210]: import numpy as np
          import pandas as pd
          from matplotlib import pyplot as plt
          from scipy.optimize import minimize, fmin_bfgs
          from sklearn.linear_model import LinearRegression
          from sklearn.metrics import mean_squared_error
          from sklearn.model_selection import train_test_split
          from tqdm import tqdm_notebook as tqdm

          %matplotlib inline
```

1 Lab 08

1.0.1 Emil Magerramov

1.1 Assignment 1

The task is to estimate mean and median statistics for the target variables y_i using Bootstrap procedure.
Then, linear models for the target variables should be created and coefficients of these models (including the intercept) should be estimated using Bootstrap.

1.1.1 Load the datasets

```
In [93]: # The datasets are swapped for some reason
         dataset2 = pd.read_csv("./ds-boot-1.csv", sep='\t')
         dataset1 = pd.read_csv("./ds-boot-2.csv", sep='\t')
```

1.1.2 Check the first dataset

```
In [213]: dataset1.head(3)

Out[213]:          p1   p2   p3       p4   p5       p6       p7       p8       p9   p10  \
          0  6.48148  3.0  5.0  7.75000  0.0  7.16667  8.16667  9.66667  6.16667   9.0
          1  5.74074  4.0  8.0  7.33333  8.0  8.83333  9.75000  9.66667  9.00000  10.0
          2  7.59259  7.0  8.0  7.66667  8.0  9.66667  9.50000  6.16667  9.66667   6.5

             ...   p19   p20      p21      p22  p23      p24      p25  p26      p27  p28
```

```
0 ...   0.0   3.0  1.66667   1.33333  0.0  1.66667  3.16667  0.0  0.00000  0.0
1 ...   7.0   3.5  8.46667   7.33333  7.0  2.50000  5.50000  5.0  8.66667  8.0
2 ...   8.0  10.0  8.50000  10.00000  9.0  3.50000  3.50000  9.0  6.50000  7.0

[3 rows x 28 columns]
```

1.1.3 Split the 1st dataset into X and y

```
In [95]: target_names = ['y' + str(i) for i in range(1, 6)]
         targets = dataset1[target_names]
         dataset1.drop(target_names, axis=1, inplace=True)
         dataset1.drop('id', axis=1, inplace=True)
```

```
In [34]: def bootstrap_estimate(func, data, B=1000):
             """
             Args:
                 func - statistic to estimate
                 data - vector of values
                 B - number of bootstrap samples

             Returns:
                 estimate, std - estimation for the func
             """
             samples = np.random.choice(data, size=(B, len(data)))
             statistics = np.apply_along_axis(func, axis=1, arr=samples)
             b_mean = np.mean(statistics)
             b_std = np.std(statistics)
             return b_mean, b_std
```

1.1.4 Estimate the targets y_i using Bootstrap

```
In [35]: for target_name in target_names:
             target = targets[target_name]
             print("Evaluating", target_name)

             print("Mean:")
             mean_b_mean, mean_b_std = bootstrap_estimate(np.mean, target)
             print("({} +- 1.96 * {}) for 95% confidence".format(mean_b_mean, mean_b_std))
             print()
             print("Median:")
             median_b_mean, median_b_std = bootstrap_estimate(np.median, target)
             print("({} +- 1.96 * {}) for 95% confidence".format(median_b_mean, median_b_std))
             print(); print()
```

```
Evaluating y1
Mean:
(4.025525 +- 1.96 * 0.46388209641567324) for 95% confidence

Median:
```

```
(4.00625 +- 1.96 * 0.573714595857557) for 95% confidence


Evaluating y2
Mean:
(5.49970625 +- 1.96 * 0.52201355911359) for 95% confidence

Median:
(6.172875 +- 1.96 * 0.7124814800224635) for 95% confidence


Evaluating y3
Mean:
(7.33241775 +- 1.96 * 0.45930786807155555) for 95% confidence

Median:
(8.309085 +- 1.96 * 0.2750371570806391) for 95% confidence


Evaluating y4
Mean:
(1.5212154999999998 +- 1.96 * 0.11332830983364219) for 95% confidence

Median:
(1.85524 +- 1.96 * 0.13428623309930174) for 95% confidence


Evaluating y5
Mean:
(5.607217 +- 1.96 * 0.4242976871089448) for 95% confidence

Median:
(6.195824999999999 +- 1.96 * 0.4247392074850167) for 95% confidence
```

```
In [100]: def bootstrap_estimate_regressor(regressor_class, X, y, B=1000):
              """
              Args:
                  regressor - class of the regressor (example: LinearRegression)
                  X - data, np.array
                  y - target, np.array
                  B - number of bootstrap samples

              Returns:
                  means, stds: tuple of lists of values
              """
```

```
    resample_indices = np.random.choice(list(range(X.shape[0])), size=(B, X.shape[0]))
    coefficients = np.ndarray((B, X.shape[1] + 1)) # + 1 for the intercept

    for i in range(B):
        X_resample = X.iloc[resample_indices[i], :]
        y_resample = y[resample_indices[i]]
        regressor = regressor_class()
        regressor.fit(X_resample, y_resample)
        coefficients[i] = np.concatenate((regressor.coef_, [regressor.intercept_]))

    coef_means, coef_stds = [], []
    for i in range(coefficients.shape[1]):
        cur_coef = coefficients[:, i]
        coef_means.append(np.mean(cur_coef))
        coef_stds.append(np.std(cur_coef))

    return coef_means, coef_stds
```

1.1.5 Create a linear model for each y_i

Create a LinearRegression model for each y_i and evaluate it using Mean Squared Error.

```
In [211]: for target_name in target_names:
              target = targets[target_name]
              X_train, X_test, y_train, y_test = train_test_split(dataset1, target, test_size=0.
              regressor = LinearRegression()
              regressor.fit(X_train, y_train)
              y_pred = regressor.predict(X_test)
              print("MSE for", target_name, "is {}".format(mean_squared_error(y_pred, y_test)))
              print()
```

```
MSE for y1 is 77.88111176274357

MSE for y2 is 43.4188974167444

MSE for y3 is 3.6396430922531744

MSE for y4 is 14.487679595848666

MSE for y5 is 4.757983590806139
```

1.1.6 Now estimate confidence intervals for the coeffifients of a linear model

For the second part of the first assignment the resampling procedure is the following:
* Sample train data from X and y with repetitions B times.
* Then, train a linear model on these data and save the coeffients of the model.

* Finally, calculate the mean and the standard deviation for each coeffifient to estimate the confidence interval.

```
In [212]: for target_name in target_names:
              target = targets[target_name]
              print("Evaluating", target_name, "confidence intervals for coeffs for 95% confiden

              coef_means, coef_stds = bootstrap_estimate_regressor(LinearRegression, dataset1, t
              for i in range(len(coef_means)):
                  current_coef_name = "coefficient_" + str(i)
                  if i == len(coef_means) - 1:
                      current_coef_name = "the intercept"
                  print("Estimating", current_coef_name)
                  print("({} +- 1.96 * {})".format(coef_means[i], coef_stds[i]))
              print()
```

```
Evaluating y1 confidence intervals for coeffs for 95% confidence
Estimating coefficient_0
(0.3052473918532328 +- 1.96 * 0.2459362485313749)
Estimating coefficient_1
(-0.016330110599094335 +- 1.96 * 0.24739534023004342)
Estimating coefficient_2
(-0.08665757880476911 +- 1.96 * 0.32958021074123506)
Estimating coefficient_3
(0.5928848760191404 +- 1.96 * 0.3360349365294616)
Estimating coefficient_4
(-0.25621866702969215 +- 1.96 * 0.3142084759283443)
Estimating coefficient_5
(0.22120323093850972 +- 1.96 * 0.4324485246556507)
Estimating coefficient_6
(-0.37903163384217897 +- 1.96 * 0.40331838047058954)
Estimating coefficient_7
(-0.08635390600581937 +- 1.96 * 0.3077645649043386)
Estimating coefficient_8
(-0.044652940855385834 +- 1.96 * 0.32766974253338427)
Estimating coefficient_9
(-0.20750107228094405 +- 1.96 * 0.30756256587419756)
Estimating coefficient_10
(0.3783408871328197 +- 1.96 * 0.3874257314125129)
Estimating coefficient_11
(0.05627876378175603 +- 1.96 * 0.3733778257325087)
Estimating coefficient_12
(0.14104371592813875 +- 1.96 * 0.32545630720889684)
Estimating coefficient_13
(-0.03368434022278277 +- 1.96 * 0.27842150580340197)
Estimating coefficient_14
(-0.43340294575512983 +- 1.96 * 0.2670942506592189)
Estimating coefficient_15
```

```
(0.19967344374691132 +- 1.96 * 0.19949887609195938)
Estimating coefficient_16
(0.32363421909956036 +- 1.96 * 0.2525227326283371)
Estimating coefficient_17
(-0.05794376611311911 +- 1.96 * 0.2486578899759727)
Estimating coefficient_18
(-0.21187147231598252 +- 1.96 * 0.2524699983418228)
Estimating coefficient_19
(-0.16111621699091042 +- 1.96 * 0.2603676032291606)
Estimating coefficient_20
(0.3609088717267041 +- 1.96 * 0.26108781239312406)
Estimating coefficient_21
(-0.018903173521867246 +- 1.96 * 0.2208717582145471)
Estimating coefficient_22
(0.34377321899604113 +- 1.96 * 0.20080065742351835)
Estimating coefficient_23
(-0.24272363459481638 +- 1.96 * 0.25358122652890114)
Estimating coefficient_24
(0.21162784257312348 +- 1.96 * 0.27878195332390077)
Estimating coefficient_25
(0.0504874395443007 +- 1.96 * 0.24696281609211213)
Estimating coefficient_26
(-0.03977049776435379 +- 1.96 * 0.28539145656239817)
Estimating coefficient_27
(-0.16115261473659842 +- 1.96 * 0.21294535618668742)
Estimating the intercept
(-0.022447894970708544 +- 1.96 * 0.43529900616369255)

Evaluating y2 confidence intervals for coeffs for 95% confidence
Estimating coefficient_0
(-0.022584535374032776 +- 1.96 * 0.4065884305569933)
Estimating coefficient_1
(0.11213123799836339 +- 1.96 * 0.35870668689750496)
Estimating coefficient_2
(-0.11165303164140478 +- 1.96 * 0.4703292646086423)
Estimating coefficient_3
(0.21008764464581062 +- 1.96 * 0.49540042517950467)
Estimating coefficient_4
(0.06198518206297909 +- 1.96 * 0.473291949106145)
Estimating coefficient_5
(0.4982033672610486 +- 1.96 * 0.7002382154229662)
Estimating coefficient_6
(0.38477984563870893 +- 1.96 * 0.4889208619478313)
Estimating coefficient_7
(-0.3800606859586509 +- 1.96 * 0.48398335418740546)
Estimating coefficient_8
(-0.06532273823765576 +- 1.96 * 0.4719263481879249)
Estimating coefficient_9
```

```
(-0.05244154298568181 +- 1.96 * 0.47441563597876846)
Estimating coefficient_10
(0.19404948631167543 +- 1.96 * 0.4910560066017214)
Estimating coefficient_11
(0.05057302930415489 +- 1.96 * 0.5950537203595693)
Estimating coefficient_12
(-0.006466221887143646 +- 1.96 * 0.4733387680947154)
Estimating coefficient_13
(0.1922931976275236 +- 1.96 * 0.3514369647208421)
Estimating coefficient_14
(-0.380733516000323 +- 1.96 * 0.4161633302038853)
Estimating coefficient_15
(0.17966669611647312 +- 1.96 * 0.2999818627292036)
Estimating coefficient_16
(0.19437599085032298 +- 1.96 * 0.3292043942104859)
Estimating coefficient_17
(0.02670114418747452 +- 1.96 * 0.3279674110842693)
Estimating coefficient_18
(-0.3485750055508314 +- 1.96 * 0.3885978533090944)
Estimating coefficient_19
(-0.15759721933204907 +- 1.96 * 0.3973456792836041)
Estimating coefficient_20
(0.33282811096363885 +- 1.96 * 0.3809260007768491)
Estimating coefficient_21
(0.2099891678993505 +- 1.96 * 0.3221576461558138)
Estimating coefficient_22
(-0.11114938735519185 +- 1.96 * 0.27730359936850785)
Estimating coefficient_23
(-0.01502622163638666 +- 1.96 * 0.4651778029292537)
Estimating coefficient_24
(0.11232364649227085 +- 1.96 * 0.42304441416552485)
Estimating coefficient_25
(0.22977252470315832 +- 1.96 * 0.3815910513793848)
Estimating coefficient_26
(-0.27930789294282227 +- 1.96 * 0.5735527114780614)
Estimating coefficient_27
(-0.19860890728850297 +- 1.96 * 0.26546193711402405)
Estimating the intercept
(-0.017610235874727733 +- 1.96 * 0.40995780424756484)

Evaluating y3 confidence intervals for coeffs for 95% confidence
Estimating coefficient_0
(0.051177439005156485 +- 1.96 * 0.07426591496685372)
Estimating coefficient_1
(0.03535417571327615 +- 1.96 * 0.056768119889512816)
Estimating coefficient_2
(0.05472680855374009 +- 1.96 * 0.07986477023652078)
Estimating coefficient_3
```

```
(0.07448686302870053 +- 1.96 * 0.08123214432321112)
Estimating coefficient_4
(0.10210684277000757 +- 1.96 * 0.07688188837102766)
Estimating coefficient_5
(0.007926945930539879 +- 1.96 * 0.10755424828131124)
Estimating coefficient_6
(0.1648399977970918 +- 1.96 * 0.0931687528325408)
Estimating coefficient_7
(0.13169291444211637 +- 1.96 * 0.09189482679800252)
Estimating coefficient_8
(0.08023732111018501 +- 1.96 * 0.10203247555275198)
Estimating coefficient_9
(0.0529842210299951 +- 1.96 * 0.07987468239909688)
Estimating coefficient_10
(0.13324413909867183 +- 1.96 * 0.09687855936999933)
Estimating coefficient_11
(0.11832614580007716 +- 1.96 * 0.10080780128655152)
Estimating coefficient_12
(0.006439010216469216 +- 1.96 * 0.07908851011075851)
Estimating coefficient_13
(0.05243882446759999 +- 1.96 * 0.08261316634112427)
Estimating coefficient_14
(0.07524449704684713 +- 1.96 * 0.06782486184870903)
Estimating coefficient_15
(-0.0017363200144281188 +- 1.96 * 0.04914942297915374)
Estimating coefficient_16
(-0.04179891891580384 +- 1.96 * 0.06784257165707387)
Estimating coefficient_17
(-0.0006534095903160185 +- 1.96 * 0.0736982023081848)
Estimating coefficient_18
(0.04753107402424646 +- 1.96 * 0.07063356766064761)
Estimating coefficient_19
(0.015284110685826345 +- 1.96 * 0.06600422946304654)
Estimating coefficient_20
(-0.012298011778592672 +- 1.96 * 0.07096772085127713)
Estimating coefficient_21
(-0.03401792013636229 +- 1.96 * 0.06607130525933012)
Estimating coefficient_22
(7.967483943773418e-05 +- 1.96 * 0.052669684968453756)
Estimating coefficient_23
(0.0398720775780162 +- 1.96 * 0.06884110503608394)
Estimating coefficient_24
(-0.07365876404514299 +- 1.96 * 0.07897643033819139)
Estimating coefficient_25
(-1.0838306214789596e-05 +- 1.96 * 0.0708041901291112)
Estimating coefficient_26
(-0.00028346969200432326 +- 1.96 * 0.07973769595338481)
Estimating coefficient_27
```

```
(-0.01917399553521352 +- 1.96 * 0.051182201456378924)
Estimating the intercept
(0.02201087888094191 +- 1.96 * 0.2485220391424638)

Evaluating y4 confidence intervals for coeffs for 95% confidence
Estimating coefficient_0
(-0.02313261304826252 +- 1.96 * 0.04289891665685881)
Estimating coefficient_1
(-0.0008685429033719769 +- 1.96 * 0.03525638938375469)
Estimating coefficient_2
(0.026510156113911908 +- 1.96 * 0.057802976408654075)
Estimating coefficient_3
(0.02831602880028164 +- 1.96 * 0.04606308464533662)
Estimating coefficient_4
(0.04434768808641603 +- 1.96 * 0.0412342627350406)
Estimating coefficient_5
(0.04968579345040043 +- 1.96 * 0.06834707497027112)
Estimating coefficient_6
(-0.020119697394872408 +- 1.96 * 0.061167106198356164)
Estimating coefficient_7
(0.06117540772141643 +- 1.96 * 0.0530168958828602)
Estimating coefficient_8
(0.042656301388637574 +- 1.96 * 0.059053668782145394)
Estimating coefficient_9
(0.02700560840307788 +- 1.96 * 0.05758894828899638)
Estimating coefficient_10
(-0.006849167270962192 +- 1.96 * 0.06497208205804852)
Estimating coefficient_11
(-0.001469518006729794 +- 1.96 * 0.06619149484394353)
Estimating coefficient_12
(-0.06371605195867522 +- 1.96 * 0.04711529808152812)
Estimating coefficient_13
(0.03729506897631613 +- 1.96 * 0.04267992253262685)
Estimating coefficient_14
(-0.0291086765869272 +- 1.96 * 0.050214947284385573)
Estimating coefficient_15
(0.024175362044688557 +- 1.96 * 0.031484685622264794)
Estimating coefficient_16
(-0.04603793474200113 +- 1.96 * 0.045012604598961815)
Estimating coefficient_17
(0.03306792903542521 +- 1.96 * 0.04078339380904264)
Estimating coefficient_18
(0.027545891900325237 +- 1.96 * 0.04015313379951429)
Estimating coefficient_19
(0.029576665867569386 +- 1.96 * 0.04692715206392505)
Estimating coefficient_20
(0.04606073701779964 +- 1.96 * 0.045845976782851856)
Estimating coefficient_21
```

```
(-0.047972514460880515 +- 1.96 * 0.043659309267359746)
Estimating coefficient_22
(0.019270540598781954 +- 1.96 * 0.03368200326002764)
Estimating coefficient_23
(0.015413525875169655 +- 1.96 * 0.04751720108814641)
Estimating coefficient_24
(-0.05519363775567555 +- 1.96 * 0.05676146944158244)
Estimating coefficient_25
(0.01741962715006592 +- 1.96 * 0.04392499699053847)
Estimating coefficient_26
(0.00850904028420664 +- 1.96 * 0.04431152193034938)
Estimating coefficient_27
(-0.0006467801122101742 +- 1.96 * 0.031162015220312374)
Estimating the intercept
(0.0017643177366053469 +- 1.96 * 0.04828330655329393)

Evaluating y5 confidence intervals for coeffs for 95% confidence
Estimating coefficient_0
(0.09663841059101225 +- 1.96 * 0.15981984763337023)
Estimating coefficient_1
(0.03499265201869525 +- 1.96 * 0.14877430039477219)
Estimating coefficient_2
(-0.03233122744435318 +- 1.96 * 0.20912687306104377)
Estimating coefficient_3
(0.3134966158989402 +- 1.96 * 0.19650934877460513)
Estimating coefficient_4
(-0.04308868686595181 +- 1.96 * 0.1699256546908766)
Estimating coefficient_5
(0.256824407168735 +- 1.96 * 0.2810546317922285)
Estimating coefficient_6
(0.025008048938897312 +- 1.96 * 0.20217867263225608)
Estimating coefficient_7
(-0.08509437382045641 +- 1.96 * 0.18358863542206175)
Estimating coefficient_8
(-0.006785705100062131 +- 1.96 * 0.23968477189752316)
Estimating coefficient_9
(-0.058725180653944616 +- 1.96 * 0.20505820031854122)
Estimating coefficient_10
(0.21950737092528733 +- 1.96 * 0.21893310634346577)
Estimating coefficient_11
(0.06505049464335926 +- 1.96 * 0.22062724921374025)
Estimating coefficient_12
(0.039742479875343566 +- 1.96 * 0.19975855698876616)
Estimating coefficient_13
(0.06602217970651464 +- 1.96 * 0.17902673634033595)
Estimating coefficient_14
(-0.2514105648040371 +- 1.96 * 0.19986283374429706)
Estimating coefficient_15
```

```
(0.12384555067052841 +- 1.96 * 0.12654246411827705)
Estimating coefficient_16
(0.13536442726897532 +- 1.96 * 0.14861600150264492)
Estimating coefficient_17
(-0.002269349962124311 +- 1.96 * 0.14369074039864907)
Estimating coefficient_18
(-0.14048577117362687 +- 1.96 * 0.19043629224108788)
Estimating coefficient_19
(-0.08665844514661092 +- 1.96 * 0.17000778979736222)
Estimating coefficient_20
(0.21533259504534066 +- 1.96 * 0.17887271265106924)
Estimating coefficient_21
(0.03406242979686218 +- 1.96 * 0.15521541148687776)
Estimating coefficient_22
(0.09516770603437226 +- 1.96 * 0.1283111869199293)
Estimating coefficient_23
(-0.08244489614024167 +- 1.96 * 0.17672305019252194)
Estimating coefficient_24
(0.06972900932301525 +- 1.96 * 0.174692240999731)
Estimating coefficient_25
(0.10481491538211628 +- 1.96 * 0.17820493459955764)
Estimating coefficient_26
(-0.08785290518016846 +- 1.96 * 0.21742986709516485)
Estimating coefficient_27
(-0.11945029852127398 +- 1.96 * 0.1258410623380885)
Estimating the intercept
(0.000526232302898955 +- 1.96 * 0.12940758969107188)
```

1.2 Assignment 2

The task is to determine the best model for the mean number of bugs as a function of time. Then, the confidence intervals for each of the model's parameters should be estimated using Bootstrap.

1.2.1 Check the 2nd dataset

In [201]:
```
dataset2.head(3)
```

Out[201]:

	day	bugsPerDay	cummBugs
0	1	0	0
1	2	1	1
2	3	0	1

In [199]:
```
def ros(t, a=1, b=1):
    """
    S-shaped model.
    Accepts t as a timepoint and a, b as model parameters.
```

```
        Returns the predicted number of bugs given timepoint.
        """
        return a*(1 - (1 + b*t)*np.exp(-b*t))
```

Since the data is a time series, the proposed approach to do the sampling is the following:
* Set random 10% of data from the `bugsPerDay` column to 0. Then, recalculate the cumulative sum.
* The sampling procedure is being repeated B times. For each sample, the least squares optimization is used to determine the best model given the data.
* Then, the mean and the standard deviation are calculated to estimate the confidence interval for each of the model's parameters.
* Finally, the optimization procedure is executed on the whole dataset in order to do the comparison.

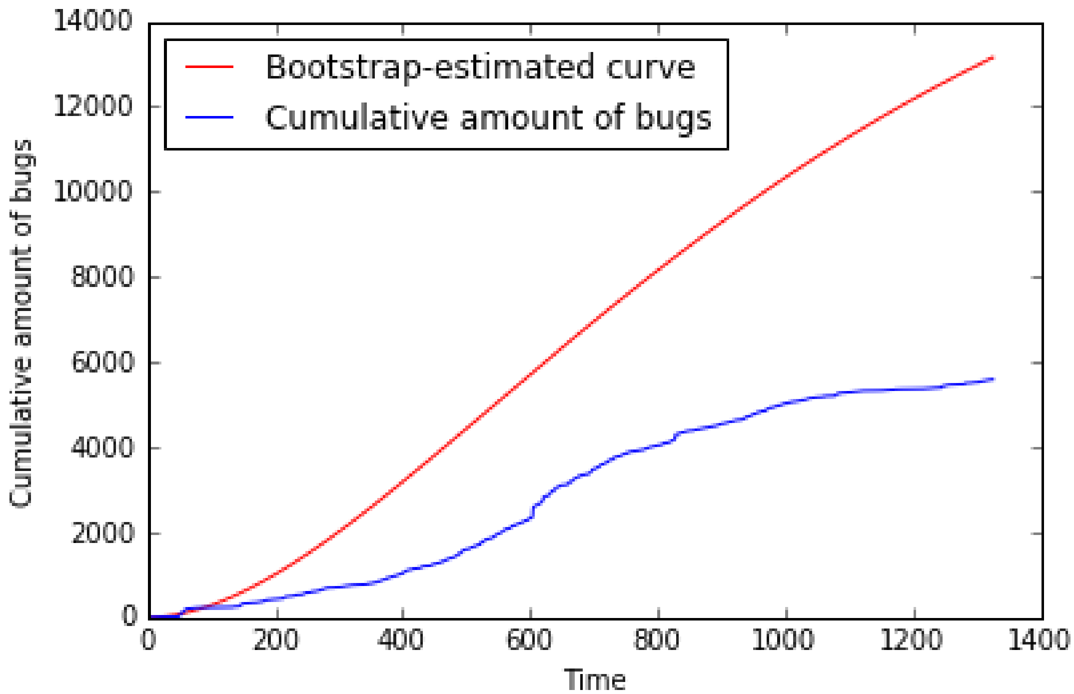
```
In [195]: def func_to_min(values):
              """
              Least squares optimization problem
              to determine the best model given data.
              """
              a, b = values
              return np.sum(np.power(np.array([ros(x, a, b) for x in np.arange(len(current_cumsu

          B = 1000
          results_a, results_b = [], []
          throw_away_inds_len = int(0.1 * len(dataset2['day']))
          for i in tqdm(range(B)):
              current_bugs = dataset2['bugsPerDay'].copy()

              # randomly set some of the bugs to 0
              throw_away_inds = np.random.choice(np.arange(len(current_bugs)), size=throw_away_i
              current_bugs[throw_away_inds] = 0
              current_cumsum = np.cumsum(current_bugs)

              # [5000., 0.001] is a starting point for the parameters
              result = minimize(func_to_min, [5000., 0.001], tol=1e-25)
              a, b = result.x

              results_a.append(a)
              results_b.append(b)

          print("a: ({} +- 1.96 * {})".format(np.mean(results_a), np.std(results_a)))
          print("b: ({} +- 1.96 * {})".format(np.mean(results_b), np.std(results_b)))
```

```
/usr/local/lib/python3.6/site-packages/scipy/optimize/optimize.py:964: RuntimeWarning: divide by
  rhok = 1.0 / (numpy.dot(yk, sk))
/usr/local/lib/python3.6/site-packages/scipy/optimize/optimize.py:964: RuntimeWarning: divide by
  rhok = 1.0 / (numpy.dot(yk, sk))
/usr/local/lib/python3.6/site-packages/scipy/optimize/optimize.py:964: RuntimeWarning: divide by
  rhok = 1.0 / (numpy.dot(yk, sk))
```

```
/usr/local/lib/python3.6/site-packages/scipy/optimize/optimize.py:964: RuntimeWarning: divide by
  rhok = 1.0 / (numpy.dot(yk, sk))
/usr/local/lib/python3.6/site-packages/scipy/optimize/optimize.py:964: RuntimeWarning: divide by
  rhok = 1.0 / (numpy.dot(yk, sk))
```

```
a: (18709.62167630103 +- 1.96 * 144827.36988481347)
b: (0.0018502726499009373 +- 1.96 * 0.00015379929476585236)
```

```
In [205]: bugs = dataset2['cummBugs']
          ros_bugs = [ros(t, np.mean(results_a), np.mean(results_b)) for t in range(len(bugs))]
          plt.plot(ros_bugs, color='r', label='Bootstrap-estimated curve')
          plt.plot(bugs, label='Cumulative amount of bugs')
          plt.xlabel("Time")
          plt.ylabel("Cumulative amount of bugs")
          plt.legend(loc='upper left');
```

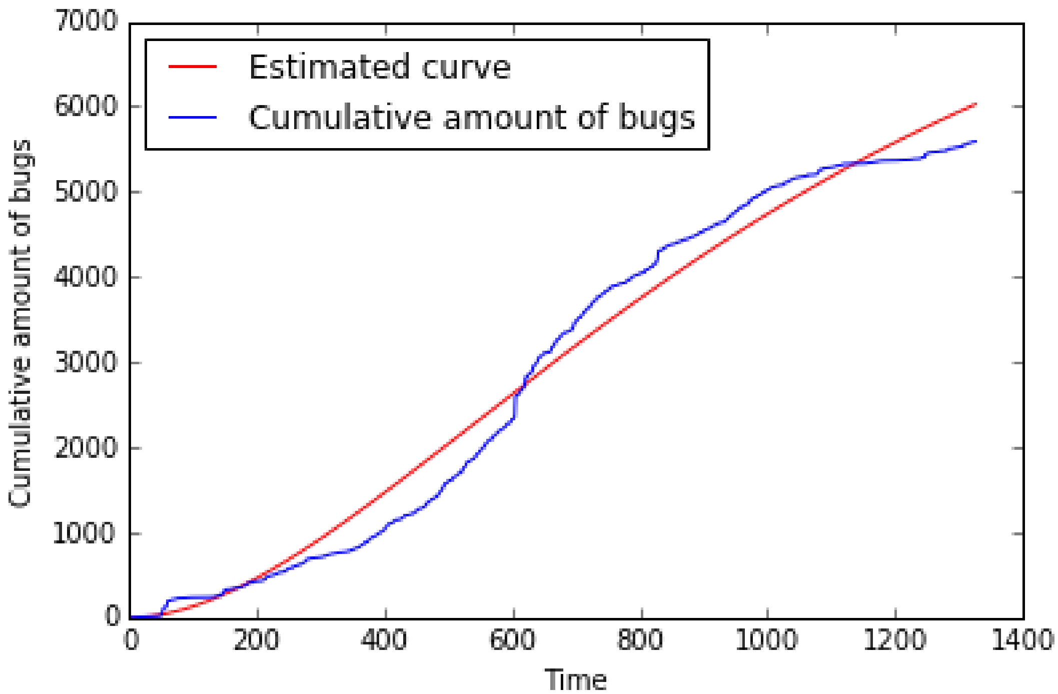


The bootstrap-estimated model pessimistically overestimated the cumulative amount of bugs over time.

1.2.2 Optimizing for the whole dataset

```
In [206]: current_cumsum = bugs
          result = minimize(func_to_min, [5000., 0.001], tol=1e-25)
```

```python
a, b = result.x
ros_bugs = [ros(t, a, b) for t in range(len(bugs))]
plt.plot(ros_bugs, color='r', label='Estimated curve')
plt.plot(bugs, label='Cumulative amount of bugs')
plt.xlabel("Time")
plt.ylabel("Cumulative amount of bugs")
plt.legend(loc='upper left');
```



The model optimized on the whole dataset fitted the data pretty good.