Simulation of Confidence Intervals on a Cubic Equation

December 30, 2023

This write-up is derived from the textbook "The Elements of Statistical Learning" from Chapter 3, exercise 3.2:

"Given data on two variables X and Y, consider fitting a cubic polynomial regression model $f(X) = \sum_{j=0}^{3} \beta_j X^j$. In addition to plotting the fitted curve, you would like a 95% confidence band about the curve. Consider the following two approaches:

(a) At each point x_0, form a 95% confidence interval for the linear function $a^T\beta = \sum_{j=0}^{3} \beta_j x_0^j$;

(b) Form a 95% confidence set for β as in (3.15), which in turn generates confidence intervals for $f(x_0)$.

How do these two approaches differ? Which band is likely to be wider? Conduct a small simulation experiment to compare the two methods."

The following code conducts said simulation experiment. First, 10 values are drawn uniformly from [0,1]. Then $y_i = 1 + x_i + 2x_i^2 + 3x_i^3 + \epsilon_i$ with ϵ_i following the $N(0, 0.5)$ distribution are generated.

[ ]:
```
import numpy as np
import plotly.graph_objects as go
from numpy.linalg import inv
from scipy.stats import chi2

n = 10
sigma = np.sqrt(0.5)

# prepare data
ones = np.ones(n)
x = np.random.uniform(0, 1, n)
x = np.sort(x)
x_square = np.square(x)
x_cubic = np.power(x, 3)

X = np.column_stack((ones, x, x_square, x_cubic))
X_T = X.transpose()

epsilon = np.random.normal(0, sigma, n)

beta = np.array([1, 1, 2, 3])
y_theory = X @ beta
```

```
y_realized = y_theory + epsilon

beta_hat = inv(X_T @ X) @ X_T @ y_realized
y_estimated = X @ beta_hat
```

[ ]:
```
# method 1
var_beta_hat = inv(X_T @ X) * (sigma**2)
tmp = X @ var_beta_hat
tmp = tmp @ X_T
width = np.diag(tmp)
width = np.sqrt(width)
width_upper = y_estimated + 1.96 * width
width_lower = y_estimated - 1.96 * width
```

Following the first method, confidence regions are generated. The variance of $\hat{y_0}$ is $Var(\hat{y_o}) = x_0(X^TX)^{-1}x_0^T$. Thus at each sample pair, the confidence interval is calculated as $\hat{y_0} \pm 1.96\sqrt{x_0(X^TX)^{-1}x_0^T}$.

[ ]:
```
# method 2
U_T = np.linalg.cholesky(X_T @ X)
U = U_T.transpose()
U_inv = inv(U)

p = 0.95
df = 4
num = 100

region_arr = []

for i in range(num):
    a = np.random.normal(0, 1, df)
    a = U_inv @ a
    a_norm = np.linalg.norm(a, ord=2)

    r = sigma * np.sqrt(chi2.ppf(p, df))
    a = a * (r/a_norm)

    beta2 = beta + a
    region = np.dot(X, beta2)
    region_arr.append(region)
```

For the second method, 100 different vectors are sampled, so 100 different $\hat{\beta}$'s are obtained.

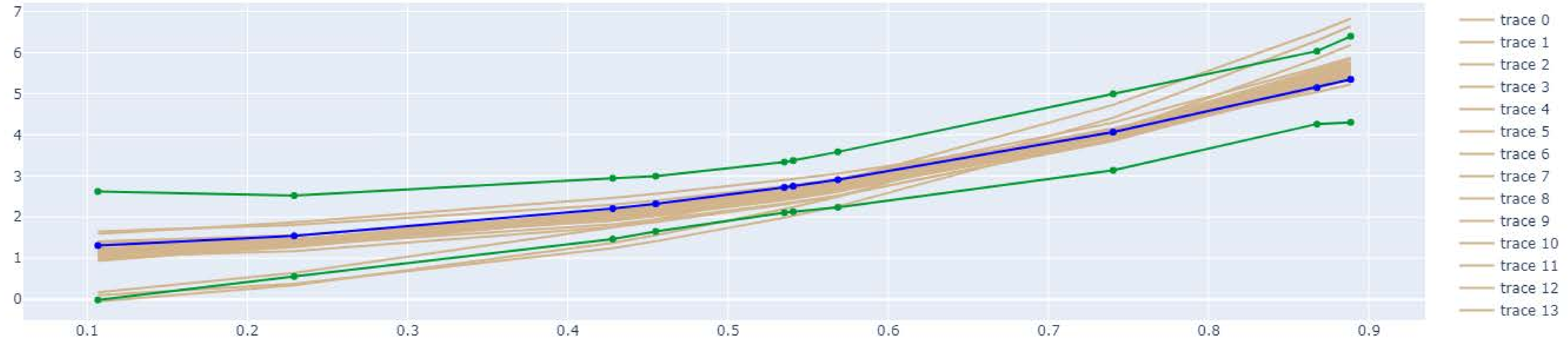
[ ]:
```
# plot
fig = go.Figure()

for i in range(num):
    fig.add_trace(go.Scatter(x=x, y=region_arr[i], mode='lines',␣
 ↪line_color='tan'))

fig.add_trace(go.Scatter(x=x, y=y_estimated, mode='lines+markers',␣
 ↪name='estimated', line_color='blue'))
fig.add_trace(go.Scatter(x=x, y=width_upper, mode='lines+markers',␣
 ↪name='upper1', line_color='#009933'))
fig.add_trace(go.Scatter(x=x, y=width_lower, mode='lines+markers',␣
 ↪name='lower1', line_color='#009933'))


fig.show()
```

The blue colored line is calculated from the estimates of beta from ordinary least squares. The two green lines denote the 95% confidence region band (from the first method), and the tan-colored lines are sampled from the boundary of the 95% confidence set (from the second method). The region from the first method is a simultaneous confidence region, while the one from the second method is an elliptical confidence region. The latter is less strict compared to the former, hence the confidence region from the first method was expected to be wider than that from the second method. As seen, the tan-colored lines are largely contained within the green lines, so the statement holds.



0
1
2
3
4
5
6
7
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
trace 0
trace 1
trace 2
trace 3
trace 4
trace 5
trace 6
trace 7
trace 8
trace 9
trace 10
trace 11
trace 12
trace 13