ISTA 331 HOMEWORK 3: CURVE FITTING

Due Thursday, 2/27, 2020, 11:59 PM

Introduction. This homework is intended to refresh your knowledge of linear regression and introduce you to more complicated curve-fitting. It is also intended to refresh your knowledge of `pandas`, `numpy`, and `matplotlib`.

Instructions. Create a module named `hw3.py`. Below is the spec for 9 functions. Implement them and upload your module to the D2L Assignments folder.

Testing. Download `hw3_test.py` and auxiliary files and put them in the same folder as your hw3.py module. Run it from the command line to see your current correctness score. Each of the first 8 functions is worth 12.5% of your correctness score. The ninth can only hurt you. You must plot the 5 curves and have a correct legend. Details like color and legend placement don't matter, but the curves must be correct. The data must be markers, not a line (see the screenshot closeup). Missing/incorrect curves/legend are each a -3 deduction from your hw grade. You can examine the test module in a text editor to understand better what your code should do. The test module is part of the spec. The test file we will use to grade your program will be different and may uncover failings in your work not evident upon testing with the provided file. Add any necessary tests to make sure your code works in all cases.

Documentation. Your module must contain a header docstring containing your name, your section leader's name, the date, ISTA 331 Hw3, and a brief summary of the module. Each function must contain a docstring. Each function docstring should include a description of the function's purpose, the name, type, and purpose of each parameter, and the type and meaning of the function's return value.

Grading. Your module will be graded on correctness, documentation, and coding style. Code should be clear and concise. You will only lose style points if your code is a real mess. Include inline comments to explain tricky lines and summarize sections of code.

Collaboration. Collaboration is allowed. You are responsible for your learning. Depending too much on others will hurt you on the tests. "Helping" others too much harms them in reality. Cite any sources or collaborators in your header docstring. Leaving this out is dishonest.

Resources. Some links that may be useful:

- `https://docs.scipy.org/doc/numpy/reference/generated/numpy.polyfit.html`
- `http://www.statsmodels.org/stable/examples/notebooks/generated/ols.html`
- `http://www.statsmodels.org/stable/examples/index.html`
- `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html`
- `http://aa.usno.navy.mil/cgi-bin/aa_rstablew.pl?ID=AA&year=2018&task=0&state=AZ&place=Tucson`
- `https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html`
- `https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html`

Function specifications.

(1) `read_frame`. This function should read the file `sunrise_sunset.csv` into a `DataFrame` with two columns for each month. The columns should be named `Jan_r, Jan_s, Feb_r, Feb_s`, ..., and contain the sunrise and sunset times for each day of the corresponding month. **Keep the data type as `str`; don't let pandas convert it to float or int**.

For reference, the upper left corner of the CSV file looks like this:

```
01,725,1730,717,1757,651,1821,612,
02,725,1731,717,1758,650,1822,611,
03,725,1732,716,1759,649,1823,609,
04,725,1732,715,1800,647,1824,608,
05,725,1733,715,1801,646,1824,607,
06,726,1734,714,1802,645,1825,606,
07,726,1735,713,1803,644,1826,604,
08,726,1736,712,1804,643,1827,603,
09,726,1736,711,1805,641,1827,602,
10,726,1737,711,1806,640,1828,601,
11,725,1738,710,1806,639,1829,559,
```

The data frame should look like this:

	Jan_r	Jan_s	Feb_r	Feb_s	Mar_r	Mar_s
1	725	1730	717.0	1757.0	651	1821
2	725	1731	717.0	1758.0	650	1822
3	725	1732	716.0	1759.0	649	1823
4	725	1732	715.0	1800.0	647	1824
5	725	1733	715.0	1801.0	646	1824
6	726	1734	714.0	1802.0	645	1825
7	726	1735	713.0	1803.0	644	1826

(2) `get_daylength_series`. This function should take the data frame produced by `read_frame` as an argument and return a `Series` containing the length of each day in the data frame, indexed from 1 to 365 (we ignore the month).

Hint/suggestion: concatenate (using `pd.concat`) the appropriate columns from the data frame into a `Series` containing all of the sunrise times, and another `Series` containing all of the sunset times. You will have to clean the data by removing `NaN`s (which occur at dates that don't exist, like Apr 31) and converting the time strings (in `hhmm` format) into raw minutes. Then subtract.

The data in the resulting `Series` should be integers.

(3) `best_fit_line`. This function takes a `Series` of day lengths as an argument, fits a linear model to it using `statsmodels.OLS`, and returns a tuple containing `results.params, results.rsquared, results.mse_resid**0.5, results.fvalue, results.f_pvalue` (here, `results` is the `RegressionResults` object returned by `statsmodels.OLS`; if you name this object something different make sure to return the right values.)

(4) `best_fit_parabola`. This function does the same thing as the previous function, except it fits a quadratic ($y = ax^2 + bx + c$) instead of a line.

(5) `best_fit_cubic`. Same as before, but this one fits a cubic ($y = ax^3 + bx^2 + cx + d$)

(6) `r_squared`. R^2, also called the coefficient of determination, is a common measure of goodness of fit, i.e. a measure of how well a model fits a collection of observed values. This function takes a `Series` and a function and returns R^2. The `Series` is the set of observations (the index is the x values and the data is the corresponding y values). The function argument is the model. `statsmodels` calculated R^2 for us in the models we have fit to our data so far; but, we can't use `statsmodels` to fit a sine curve, so we are going to write a function to calculate it ourselves.

Use the following ingredients in the calculation:

- The total sum of squares (proportional to the sample variance of y)

$$\mathrm{SS}_{\mathrm{tot}} = \sum_{i=1}^{N} (y_i - \bar{y})^2$$

- The model sum of squares:

$$\mathrm{SS}_{\mathrm{model}} = \sum_{i=1}^{N} (\hat{y}_i - \bar{y})^2$$

(remember $\hat{y}_i$ is the predicted value of y at $x = x_i$)

- The sum of squares of residuals:

$$\mathrm{SS}_{\mathrm{res}} = \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

- Finally:

$$R^2 = 1 - \frac{\mathrm{SS}_{\mathrm{res}}}{\mathrm{SS}_{\mathrm{tot}}}$$

(7) `best_fit_sine`. This one fits a sine with form $y = a\sin(bx + c) + d$. As we saw in class, we can't use `statsmodels` to do this directly because two of the parameters that we want to optimize are not coefficients and OLS is only good for coefficients. So we will use `scipy.optimize.curve_fit()`. You may refer to the Jupyter notebook `curve_fitting_2.ipynb` for an example; the application of `curve_fit` there is very similar.

In addition to the required arguments, we need to pass in starting estimates for `[a, b, c, d]`. Otherwise, `curve_fit` will start them all as 1 by default, and that is too far away from the correct values for the optimization to find the best solution. You will have to examine the data to choose reasonable starting estimates. They don't have to be perfect, just vaguely close.

`curve_fit` returns two values. The first one is the optimized parameters. If this value is stored in a variable called `popt`, use this syntax to define a function to pass to your `r_squared` function:

```
f = lambda x: popt[0] * np.sin(popt[1] * x + popt[2]) + popt[3]
```

Lambda's are one-line, "anonymous" functions.[1] They are very handy in situations like this. Alternatively, you can define a model function explicitly, similar to what is done in the in-cass notebook. Return all of the same info as in the previous model derivation functions. For the F-statistic and the F-statistic p-value, return 813774.14839414635 and 0.0, respectively.

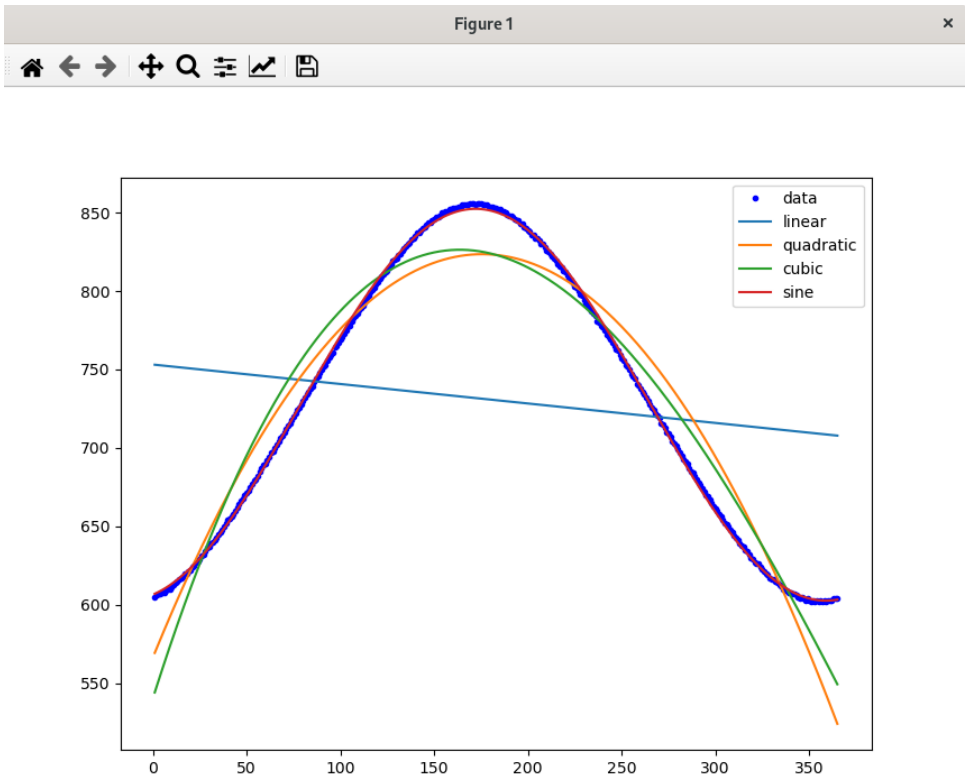
(8) `get_results_frame` This function takes a daylength Series and returns this a data frame containing the coefficients, R^2, RMSE, F-statistic, and ANOVA p-value for each of the four models above. The frame should look like this:

	a	b	c	d	R^2	RMSE	F-stat	F-pval
Linear	-0.124168	753.152913	NaN	NaN	0.022445	86.579941	8.334511	4.122740e-03
Quadratic	-0.008344	2.929903	566.345579	NaN	0.922712	24.378128	2160.904539	5.594986e-202
Cubic	0.000011	-0.014133	3.778522	540.28580	0.935022	22.383496	1731.589098	7.427419e-214
Sine	124.934633	0.016813	-1.322029	727.50444	0.999554	1.854176	813774.148394	0.000000e+00

(9) `make_plot` This function takes a daylength `Series` and a `results` frame and creates the following image (the second image is a closeup of part of the first, to illustrate that the data should be plotted as points and the fitted models as curves). Your function should end with `plt.show()` (this ensures that the plot is displayed on the screen).

The plot should look something like this:

[1] This one isn't really "anonymous" because we assigned it a name, but we used the same syntax.



Zooming in, you should be able to see that the data are plotted as points and the models as curves: