Exercise 1: Polynomial Regression

In this exercise our aim is to check what happens when we try to learn from data in a simple setting.

We do this by using polynomial functions.

Step 1: Define the hidden functions

First of all we generate the data from *hidden* functions: given a certain analytical function, we generate a subset of (x, y) tuples from it, and we take them as the starting ground for our exercise.

We will generate data from two *hidden* functions:

$$f_A(x) = 2x,$$

and

$$f_B(x) = 2x - 10x^5 + 15x^10.$$

Step 2: Generate data from the hidden functions

We will generate 10 tuples. Since we want them to be the *exact* function, we will generate them from the general function

$$y_\mu = f(x_\mu) + \eta_\mu$$

where the η_μ is always identically zero (i.e. mean zero and variance zero).

We define a function `get_dataset` in order to get tuples from the above formula, given a domain of the x values, the number of tuples, and the mean and variance of the noise term η. It returns the x and y arrays, depending on the `func_name` name of the function generating the dataset (i.e. A and B in this case).

Step 3: Fit the data

The following function `get_polyfunc` yields the array of polynomial coefficients, given a degree of the polynomial and the name of the generating function. `get_polyxy` yields instead the tuple containing the x and y sample values of the polynomial function for plotting purposes with, say, 100 points, given the polynomial degree and the name function.

Step 4: Generate the test dataset

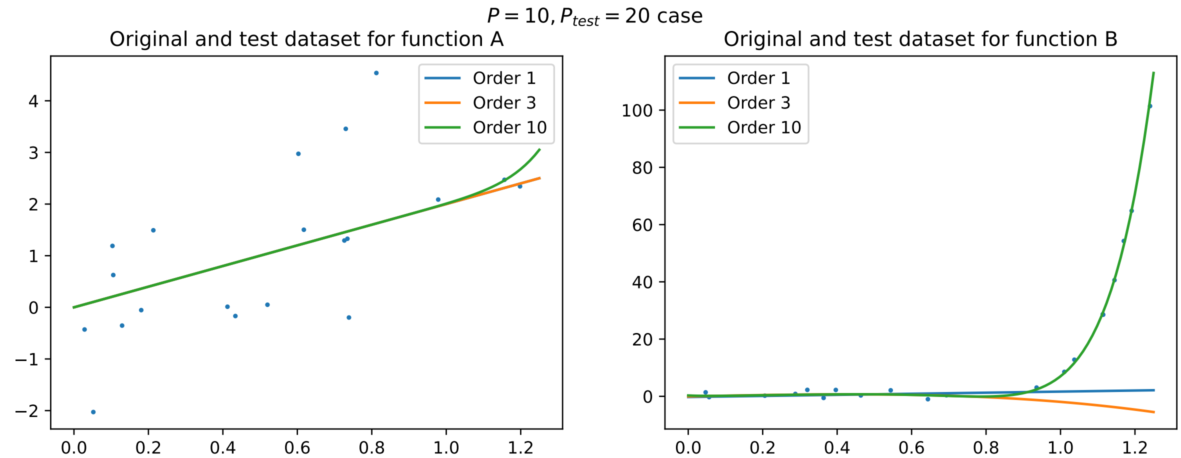
We now generate the shape of the polynomial (which we supposedly know only by *observation* and our goal is to infer the model).

We generate the polynomials in the $[0, 1.25]$ interval so that we can compare them to the test data.

We do this for each of the two generating functions, `A` and `B` and call the values `x_c` and `y_c`. We also generate the so-called *test* dataset (`x_t` vs. `y_t`).

```
/usr/lib/python3.11/site-packages/numpy/polynomial/polynomial.py:1362: RankWarning: The fit m
  return pu._fit(polyvander, x, y, deg, rcond, full, w)
/usr/lib/python3.11/site-packages/numpy/polynomial/polynomial.py:1362: RankWarning: The fit m
  return pu._fit(polyvander, x, y, deg, rcond, full, w)
```

```
{1: {'A': array([2.22044605e-16, 2.00000000e+00]),
  'B': array([-0.23620271,  1.89080077])},
 3: {'A': array([-1.44328993e-15,  2.00000000e+00, -5.45787236e-15]),
  'B': array([-0.02137902,  2.54404772, -0.15122198, -4.30649322])},
 10: {'A': array([ 9.90123024e-06,  1.99922707e+00,  1.94923219e-02, -2.39427267e-01,
          1.66940555e+00, -7.10048497e+00,  1.90495193e+01, -3.23490170e+01,
          3.36950732e+01, -1.96292419e+01,  4.89384411e+00]),
  'B': array([ 3.05986679e-01, -6.37876041e+00,  9.18655814e+01, -5.48028099e+02,
          2.00440808e+03, -4.76224365e+03,  7.46233839e+03, -7.71658955e+03,
          5.05680932e+03, -1.90483894e+03,  3.29352077e+02])}}
```



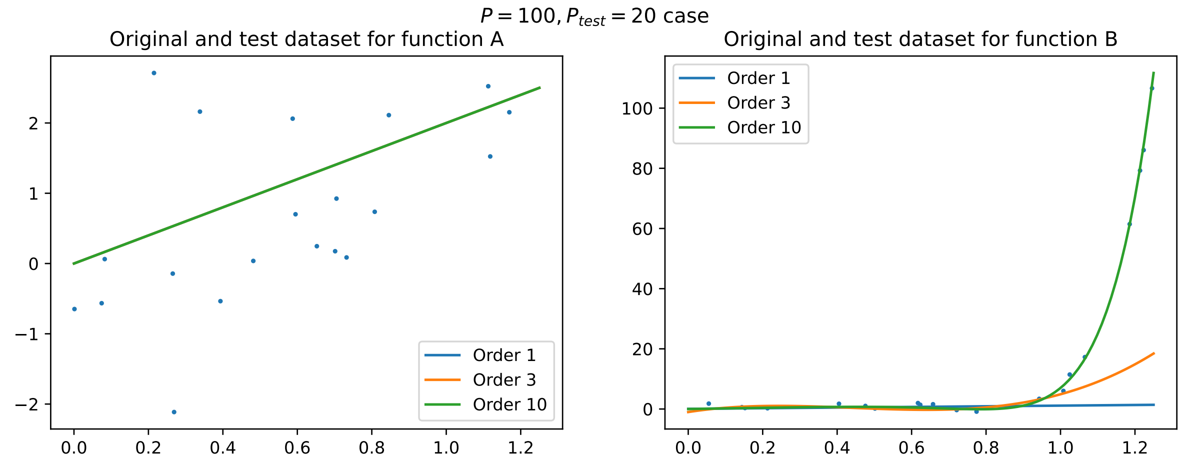
Step 5 and 6: How well does the fit describe the test dataset?

We can comment on the previous figures:

- For **function A**, order 1 and order 3 polynomial behave well (are nearly linear) in the whole domain. Order 10 instead diverges from the linear behaviour for values of $x > 1.0$, giving us a hint that probably the presence of high order terms for a dataset which is originally of lower order yields to an overfitting behaviour.
- Instead for **function B** it is the very polynomial of order 10 that well explains data points that are *out-of-sample* (i.e. for $x > 1.0$), while polynomials of order 3 and 1 poorly perform, especially in that domain.

Step 7: Let's check what happens for a bigger original dataset generating the polynomials

```
{1: {'A': array([4.4408921e-16, 2.0000000e+00]),
  'B': array([0.05281722, 1.0729709 ])},
 3: {'A': array([-6.66133815e-16,  2.00000000e+00, -1.00852932e-14,  9.27586009e-15]),
  'B': array([ -1.00754465,  19.37369394, -55.12569327,  41.66148662])},
 10: {'A': array([ 1.33226763e-15,  2.00000000e+00,  6.79618339e-13, -7.06365390e-12,
          3.80277591e-11, -1.22090296e-10,  2.46148193e-10, -3.14161613e-10,
          2.46348306e-10, -1.08330646e-10,  2.04668264e-11]),
  'B': array([-1.88737914e-15,  2.00000000e+00,  3.49653639e-12, -4.17408330e-11,
          2.69394285e-10, -1.00000000e+01,  2.39512798e-09, -3.49007756e-09,
          3.08445403e-09, -1.51248791e-09,  1.50000000e+01])}}
```

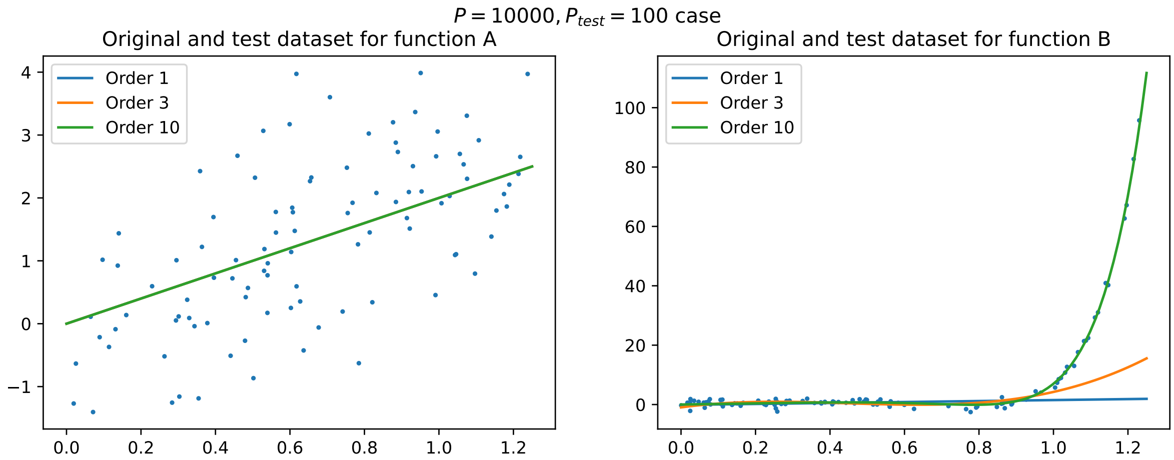


Step 8: How this affects the results?

For the $P = 100, P_{test} = 20$ case the **function B** plot doesn't seem to have changed much. This may be due to the fact that a higher number of points to generate the fits have killed higher order terms. Instead, the **function A** has seen the Order 10 polynomial *straightened up*. This may be explained by the fact that, for polynomials of order 1 and 3 much hasn't changed since they are by construction very out of the way to fitting such a nonlinear function (with finite order 5 and order 10 terms). The polynomial of order 10 instead had previously already enough information to *catch the signal*.

Step 9: Let's check, instead, for the case of a much bigger original dataset, and a bigger test dataset as well

```
{1: {'A': array([4.4408921e-16, 2.0000000e+00]),
  'B': array([-0.13180274,  1.63584754])},
 3: {'A': array([-6.66133815e-16,  2.00000000e+00,  5.98513132e-16,  2.65661240e-17]),
  'B': array([ -0.95869315,  17.12850722, -47.07657224,  35.15246417])},
 10: {'A': array([ 6.21724894e-15,  2.00000000e+00,  6.65037267e-13, -2.91068328e-12,
          7.08401386e-12, -8.45875802e-12,  9.98181934e-13,  9.91113976e-12,
         -1.18270111e-11,  5.45028660e-12, -8.25319387e-13]),
  'B': array([ 7.10542736e-15,  2.00000000e+00,  1.30562228e-12, -7.79376563e-12,
          2.80131474e-11, -1.00000000e+01,  8.96653862e-11, -7.92692578e-11,
          4.08730827e-11, -1.05329079e-11,  1.50000000e+01])}}
```



Step 10: What happens when the dataset is larger?

When the dataset is larger we have more data points, which means that our predicted models have a stronger predictive power. The robustness of the predictions can be seen from the behaviour of the *out-of-sample* polynomial, which is consistent with the previous case as well.