# Tutorial 4: Regression; optimisation

## Linear regression and least squares

1. **Quiz question:** Answer the following:
   (a) What's the difference between regression and classification?
   (b) Given a dataset of images, the task is to determine if a face is present in the image or not. Is this regression or classification?
   (c) Given a set of measurements from an array of sensors from industrial machinery, the task is to estimate of the remaining time of the machine before next maintenance cycle is due. Is this regression or classification?
   (d) Given a set of true or false answers for variety of symptoms, the task is to estimate the probability of patient having a flue? Is this regression or classification?

2. Download `linregression.py`, `t4dataset1.npz` and `t4dataset2.npz` from the Tutorial 4 website. Start Canopy, click on 'Editor', then open `linregression.py`. This is an example of linear regression that computes model parameters using the closed form least squares solution. Read through the code and comments to make sure you understand what it does. Here is a general description:

   - The script starts off by reading data from (`t4dataset1.npz`, it contains a data sample with one-dimensional array `x` and corresponding target output `ytilde`. The data is split into a training (80% of the points) and test (the remaining 20% of the points) sets.
   - The script provides functions for two flavours of the linear model of the form $y = \sum_{j=0}^{k} w_j f_j(x)$. The two flavours differ by the computation of their base functions:
     - **polynomial** model with $f_j(x) = x^k$: the evaluation of bases from input is computed with the function `polynomial_bases(x,k)`, where $k$ is the polynomial degree, the output of the model with the function `polynomial_function(x,w)`
     - and **cosines** model with $f_j(x) = cos(\frac{jx}{\pi})$: the evaluation of bases from input is computed with the function `cosines_bases(x,k)`, the output of the model with the function `cosines_function(x,w)`
   - The script implements `linearLS_params(F,ytilde)` function, which computes the parameters of the linear model for given data sample. The

function is generic to any linear model, and so the input must be provided via matrix `F` that is the evaluation of that input over the set of the base functions. The desired output is provided via the `ytilde` parameter.

- In the script, the parameters for the polynomial and cosines models are computed, printed with the corresponding root mean squared train and test error, and the data is plotted with the derived functions shown for each model.

(a) The degree of the polynomial is defined via the `k_poly` variable, and the number of cosines via `k_cos`. By default they are set to 1. Run the script - both models will give a pretty poor fit. Increase the `k_poly` and `k_cos` until you are satisfied you're getting a good fit for both models. Remember, you want simplest model (i.e. smallest `k`) that is reasonably consistent and gives good test error. The train and test errors are printed along with the parameter values. You can also visual inspect the fit. Once you're happy you're getting a good fit, record the `k_poly` and `k_cos` values and the corresponding parameter values.

(b) Change the line that reads `"t4dataset1.npz"` dataset to use the `"t4dataset2.npz"`. Repeat the exercise to find best `k_poly` and `k_cos` that results in good fit of each model to the new data. Don't bother writing down the parameters - this data is a bit more complex and so will be the models. Just find the best value for `k_poly` and `k_cos`.

# Optimisation

3. Answer the following:
   (a) What is the difference between a global and a local minimum?
   (b) What is the function of the learning rate parameter when updating the model parameters during training?
4. In Canopy's 'Editor' open `linoptimisation.py` (from Tutorial 4 website). This is an example of optimisation using steepest gradient descent for the same datasets and linear models that were used in the previous exercise. Read through the code and comments to make sure you understand what it does. Here is a general description:
   - The script reads `t4dataset1.npz` and it splits it into training and test sets.
   - The script provides the same base and model output functions for `polynomial` and `cosines` models that were used in the previous exercise.
   - The script implements `optimise_lin(F,ytilde,w,_alpha,_nEpochs)` function iterating over `_nEpochs` epochs, starting with initial parameter values `w`, and doing an update based on negative gradient using learning rate `_alpha`. No need

to compute model specific derivatives, since for a liner model $\frac{\partial J}{\partial w_j} = \sum_i f_j(x_i)(y_i - \tilde{y}_i)$ regardless of the form that $f_j(x)$ takes. To make this function generic for any linear model, the input needs to be provided after evaluation over the base functions.

- Optimisation will terminate at a point when the next update does not reduce the overall cost. The RMS value for each epoch of training is plotted after optimisation along with the visualisation of the fit over the data. The parameters found and RMS train and test errors are printed in the console.

(a) You can control which model you're optimising over with the `using_poly` variable. From the previous exercise, select the model that you think works best for the `t4dataset1.npz` dataset, and just work with that model - setting `using_poly=True` for polynomial model and `using_poly=False` for the cosines model. Set the model complexity `k` to the value that you found that worked best in the previous exercise. Run the optimisation.

(b) Does the optimisation find the same parameter values as you found in the previous exercise? Adjust the number of epochs via the `nEpoch` variable to prolong or decrease the training time. Adjust the learning rate via the `alpha` variable to increase/decrease the size of each update. Try to get the optimisation to produce parameters that are close to the optimal values found in the previous exercise in as few iterations as possible. (Hint: If the optimisation keeps terminating early, it probably means your `alpha` is too big).

# Non-linear regression

5. In Canopy's 'Editor' open `optimisation.py` (from Tutorial 4 website). This is an example of optimisation using steepest gradient descent over a non-linear model $y = w_0 \cos\Big(w_1(x - w_2)\Big) + w_3 \cos\Big(w_4(x - w_5)\Big)$. Read through the code and comments to make sure you understand what it does. Here is a general description:
   - The script reads `t4dataset1.npz` and it splits it into training and test sets.
   - The script provides `coscos_function` which computes the output of the non-linear model $y = w_0 \cos\Big(w_1(x - w_2)\Big) + w_3 \cos\Big(w_4(x - w_5)\Big)$
   - The script provides `coscos_function_derivatives` which computes partial derivatives of the coscos function with respect to its parameters. The partial derivatives for the first three parameters are as follows (the other three follow the same pattern, since they are used in the same non-linearity):
     $\partial y/\partial w_0 = \cos\Big(w_1(x - w_2)\Big)$
     $\partial y/\partial w_1 = -(x - w_2)\sin\Big(w_1(x - w_2)\Big)$
     $\partial y/\partial w_2 = w_1 \sin\Big(w_1(x - w_2)\Big)$
   - Optimisation will terminate at the point when the next update does not reduce the overall cost. The RMS value for each epoch of training is plotted after

optimisation along with the visualisation of the fit over the data. The parameters found and RMS train and test errors are printed in the console.

(a) In this exercise you will be optimising using just one non-linear model. However, the initial value of the parameters will be different every time you run the script (it's chosen at random). Run the script a few times and see what results you get. It might not be necessary, but if you want, you can adjust the `nEpoch` variable to prolong or decrease the training time, the learning rate via the `alpha` variable to increase/decrease the size of each update, and the `winit_var` to change the variance of the initial parameter choices. Keep running the script until you get a decent fit.

(b) Try the same non-linear model on the second dataset `t4dataset2.npz`.

6. **Quiz question** Based on the all the practical exercises from this lab, what are the advantages/disadvantages of working with non-linear models as opposed to linear ones?