Summary of the Internship
From Molecular Simulation to Machine Learning: Scientific Programming and Computational Chemistry

Yilin YE *†

June 11, 2020

Abstract

From Feb. 02 to Feb. 06, an internship launched by Theoretical Group of ENS, called "From Molecular Simulation to Machine Lerning: Scientific Programming and Computational Chemistry". During the four-day experience, four molecules were selected to calculate their orbital energies with the help of the application "Orca", with BLYP method and B3LYP method respectively, and then IR spectrum, as well as positions of HOMO and LUMO. Furthermore, based on an existing model of charged Lennard-Jones spheres, the machine learning simulation was engaged to fit hydration free energies, including polynomial regression, LASSO regularization, and then neural network model.

Contents

*Département de Chimie, École Normale Supérieure de Paris, 75005, France
†Contact: yyl.libri@gmail.com, yye@clipper.ens.psl.eu

1 Quantum Chemistry and Calculation

As the basis of quantum chemistry, Schrödinger equation could furnish energies of related orbitals, [1] including the kinetic energy and the potential energy, with the form below:

$$\hat{H}\Psi = E\Psi$$

$$\hat{H} = \hat{T}_n + \hat{T}_e + \hat{V}_{ne} + \hat{V}_{ee} + \hat{V}_{ne}$$

Herein, $\hat{H}$ is Hamilton operator, $\hat{T}$ is kinetic operator, and $\hat{V}$ is potential operator. Also, n represents the nuclei, as well as e for electrons. With the atomic units, we have the further expression of Hamiltonian:

$$\hat{H} = -\sum_{i=1}^{N}\frac{1}{2}\nabla_i - \sum_{A=1}^{M}\frac{1}{2M_i} - \sum_{i=1}^{N}\sum_{A=1}^{M}\frac{Z_A}{r_{iA}} + \sum_{i=1}^{N}\sum_{j>i}^{N}\frac{1}{r_{ij}} + \sum_{A=1}^{M}\sum_{B>A}^{M}\frac{Z_A Z_B}{R_{AB}}$$

For the sake of the simpler expression, the movement of nuclei $\hat{T}_n$ would be neglected in general, addressed as *Born-Oppenheimer approximation*. [2] However, it still remains a sticky problem to deal with the repulsion among electrons, and the attraction between nuclei and one electron as well. Based on *Hartree-Fock method* (HF), [3][4][5] approaches including *Self-Consistent Field* (SCF) [6] basically, *Coupled-Cluster method* (CC), [7] *Møller-Plesset Perturbation Theory* (MPn), [8] *Configuration Interaction* (CI), and related modifications generated gradually.

1.1 Density Functional Theory

Regarding that the properties of a many-electron system can be determined by functionals, namely the function of spatially dependent electron density here, *Density functional theory* (DFT) is a computational quantum mechanical modelling method to investigate the electronic structure of many-body systems, in particular atoms, molecules, and the condensed phases. Wildly popular in computational chemistry, DFT focuses on the electron density rather than wavefunctions as variables, only depending on 3 coordinates instead of $3N$ ones for wavefunctions, which could simplifier the calculation in some degree with lower cost compared with others such as HF.

As the core concept of DFT, *Hohenberg–Kohn theorems* relate to any system consisting of electrons moving under the influence of an external potential.

Theorem 1. The external potential (and hence the total energy), is a unique functional of the electron density.
If two systems of electrons, one trapped in a potential $v_1(\mathbf{r})$ and the other in $v_2(\mathbf{r})$, have the same ground-state density $v_1(\mathbf{r}) - v_2(\mathbf{r})$ is necessarily a constant.

Corollary: the ground state density uniquely determines the potential and thus all properties of the system, including the many-body wavefunction. In particular, the H–K functional, defined as $F[n] = T[n] + U[n]$, is a universal functional of the density (not depending explicitly on the external potential).

Theorem 2. The functional that delivers the ground state energy of the system gives the lowest energy if and only if the input density is the true ground state density. For any positive integer N and potential $v(\mathbf{r})$, a density functional $F[n]$ exists such that

$$E_{(v,N)}[n] = F[n] + \int v(\mathbf{r})n(\mathbf{r})\mathrm{d}^3\mathbf{r}$$

obtains its minimal value at the ground-state density of N electrons in the potential $v(\mathbf{r})$. The minimal value of is then the ground state energy of this system.

In the *Local Density Approximation* (LDA), it is assumed that the density locally could be treated as a uniform electron gas, or equivalently that the density is a slowly varying function. The exchange energy for a uniform electron gas is given by:

$$E_{\mathrm{x}}^{\mathrm{LDA}}[\rho] = -C_{\mathrm{x}}\int \rho^{4/3}(\mathbf{r})\mathrm{d}\mathbf{r}$$

$$\epsilon_{\mathrm{x}}^{\mathrm{LDA}} = -C_{\mathrm{x}}\rho^{1/3}$$

In the more general case, LDA has been virtually abandoned and replaced by the *Local Spin Density Approximation* (LSDA), which is given as the sum of the individual densities raised to the 4/3 power:

$$E_{\mathrm{x}}^{\mathrm{LSDA}}[\rho] = -2^{1/3}C_{\mathrm{x}}\int (\rho_\alpha^{4/3} + \rho_\beta^{4/3})\mathrm{d}\mathbf{r}$$

Improvements over the LSDA approach must consider a non-uniform electron gas. A step in this direction is to make the exchange and correlation energies dependent not only on the electron density but also on derivatives of the density. In the *Generalized Gradient Approximation* (GGA) methods, the first derivative of the density is included as a variable, and in addition it is required that the Fermi and Coulomb holes integrate to the required values of 1 and 0. One of the earliest and most popular GGA exchange functionals was proposed by A.D. Becke (B or B88) as a correction to the LSDA exchange energy: [9]

$$\epsilon_{\mathrm{x}}^{\mathrm{B88}} = \epsilon_{\mathrm{x}}^{\mathrm{LDA}} + \Delta\epsilon_{\mathrm{x}}^{\mathrm{B88}} = -\beta\rho^{1/3}\frac{(|\nabla\rho|/\rho^{4/3})^2}{1+6\beta(|\nabla\rho|/\rho^{4/3})\sinh^{-1}(|\nabla\rho|/\rho^{4/3})}$$

There have similarly been various GGA functionals proposed for the correlation energy. One popular functional is due to Lee, Yang and Parr (LYP),[10][11] which has the rather intimidating form shown by

$$\begin{aligned}\epsilon_c^{\mathrm{LYP}} =& -4a\frac{\rho_\alpha\rho_\beta}{\rho^2(1+d\rho^{-1/3})} - ab\omega\{\frac{\rho_\alpha\rho_\beta}{18}[144(2^{2/3})C_F(\rho_\alpha^{8/3}+\rho_\beta^{8/3}) + (47-7\delta)|\nabla\rho|^2 \\ &- (45-\delta)(|\nabla\rho_\alpha|^2+|\nabla\rho_\beta|^2) + 2\rho^{-1}(11-\delta)(\rho_\alpha|\nabla\rho_\alpha|^2+\rho_\beta|\nabla\rho_\beta|^2)] \\ &+ \frac{2}{3}\rho^2(|\nabla\rho_\alpha|^2+|\nabla\rho_\beta|^2-|\nabla\rho|^2) - (\rho_\alpha^2|\nabla\rho_\beta|^2+\rho_\beta^2|\nabla\rho_\alpha|^2)\}\end{aligned}$$

$$\omega = \frac{e^{-c\rho^{-1/3}}}{\rho^{14/3}(1+d\rho^{-1/3})}$$

$$\delta = c\rho^{-1/3} + \frac{d\rho^{-1/3}}{1+d\rho^{-1/3}}$$

The a, b, c and d parameters are determined by fitting to data for the helium atom. The LYP correlation functional is often combined with the B88 (or OPTX) exchange functional to produce the BLYP (and OLYP). In addition, consisting of B88 and HF exchange, and the LYP correlation functional, the B3LYP functional contains three parameters: [12][13]

$$E_{\mathrm{xc}}^{\mathrm{B3LYP}} = (1-a)E_{\mathrm{x}}^{\mathrm{LSDA}} + aE_{\mathrm{x}}^{\mathrm{HF}} + b\Delta E_{\mathrm{x}}^{\mathrm{B88}} + (1-c)E_{\mathrm{c}}^{\mathrm{LSDA}} + cE_{\mathrm{c}}^{\mathrm{LYP}}$$

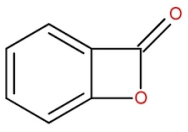
a, b, c are determined by fitting to experimental data and $a = 0.20$, $b = 0.72$ and $c = 0.81$ for B3LYP.

One of the approximations inherent essentially all *ab initio* methods is the introduction of a basis set. The type of basis functions used also influence the accuracy, and also the computational effort of *ab initio* methods scales formally as the least M_{basis}^4, attaching great importance to choose proper basis. The *Karlsruhe* basis sets have been used extensively for both HF/DFT and electron correlation methods, since they are computationally quite efficient and furthermore available for a large fraction of atoms in the periodic table. The group centered around R. Ahlrichs has designed basis sets of DZ, TZ and QZ quality for the elements up to Kr, while the *Triple Zeta Valence* (TZV) basis set is a [5s3p] contraction of an (11s6p) set of primitive functions (contraction 6,2,1,1,1 and 4,1,1). [14][15] In some cases, there would be a "P" after the basis name, indicating new *polarization* functions were added.

1.2 Orca and Practice

Developed by the Frank Neese group, ORCA [16][17] is an ab initio quantum chemistry program package that contains modern electronic structure methods including density functional theory, many-body perturbation, coupled cluster, multireference methods, and semi-empirical quantum chemistry methods. Its main field of application is larger molecules, transition metal complexes, and their spectroscopic properties.

Herein, we selected four small molecules, namely acetic acid, benzene, cyclohexane, and one benzolactone with the below skeleton, in order to simulate their energies and corresponding vibrational frequencies. Later, we would address this benzolactone as lactone simply.



However, before the formal calculation, pre-calculation should be taken into consideration with the help of Avogadro.app. [18] After drawing each structure inside, we could utilise the internal "Auto Optimization Tool" to obtain one construction with rather low energy then one .xyz file, which could be treated as the input file afterwards. First of all, we exploit BLYP to calculate energies for each molecule, with the code below:

```
| vi orca_job_1.inp
| > ! RKS BLYP D3 def2-TZVP Opt
| >
| > * xyzfile 0 1 input.xyz
|
| orca orca_job_1.inp > orca_job_1.out
```

As a matter of fact, BLYP would give out better results than Avogadro.app, but less accurate ones than B3LYP. Obviously, we could not use B3LYP directly dut to more numbers of basis set and then much longer time. Therefore, we make B3LYP calculation as the following work, with BLYP output .xyz file above as the next input file .

```
| vi orca_job_2.inp
| > ! RKS B3LYP D3 def2-TZVP Opt
| >
| > * xyzfile 0 1 orca_job_1.xyz
|
| orca orca_job_2.inp > orca_job_2.out
```

Finally, Orca provided a series of energy data optimized, playing a role as the input file to simulate the IR soon.

2 IR Spectroscopy

Infrared radiation (IR), also called infrared light sometimes, is the electromagnetic radiation with wavelengths longer than those of visible light. The infrared portion of the electromagnetic spectrum is usually divided into three regions; the near-, mid- and far- infrared, named for their relation to the visible spectrum. In general, it's invisible to the human eye, with the wavelength from 700nm to 1mm, and the photon energy from 1.7 ev to 1.24 meV. Discovered in 1800 by astronomer Sir William Herschel, Infrared radiation could be emitted or absorbed by molecules when they change their rotational-vibrational movements. [19]

Infrared spectroscopy (IR spectroscopy or vibrational spectroscopy) involves the interaction of infrared radiation with matter. It excites vibrational modes in a molecule through a change in the dipole moment, making it a useful frequency range for study of these energy states for molecules of the proper symmetry, as each vibration mode would furnish one corresponding absorption shown on the spectrum. Therefore, for some vibration modes without any change of the dipole moment, there would be no signal seen in IR spectroscopy. Considering one molecule inside an external electric field with the intensity ϵ, we could obtain the energy E as a Taylor series below:

$$E(\epsilon) = E(\epsilon = 0) + \epsilon \frac{\mathrm{d}E}{\mathrm{d}\epsilon}|_{\epsilon=0} + \frac{1}{2}\epsilon^2 \frac{\mathrm{d}^2 E}{\mathrm{d}\epsilon^2}|_{\epsilon=0} + \cdots$$

Thus we could extract the dipole moment as $\mu = -\frac{\mathrm{d}E}{\mathrm{d}\epsilon}|_{\epsilon=0}$, and the infrared intensities relates to $\frac{\mathrm{d}^2 E}{\mathrm{d}x_i \mathrm{d}\epsilon}$.

2.1 Calculation Results

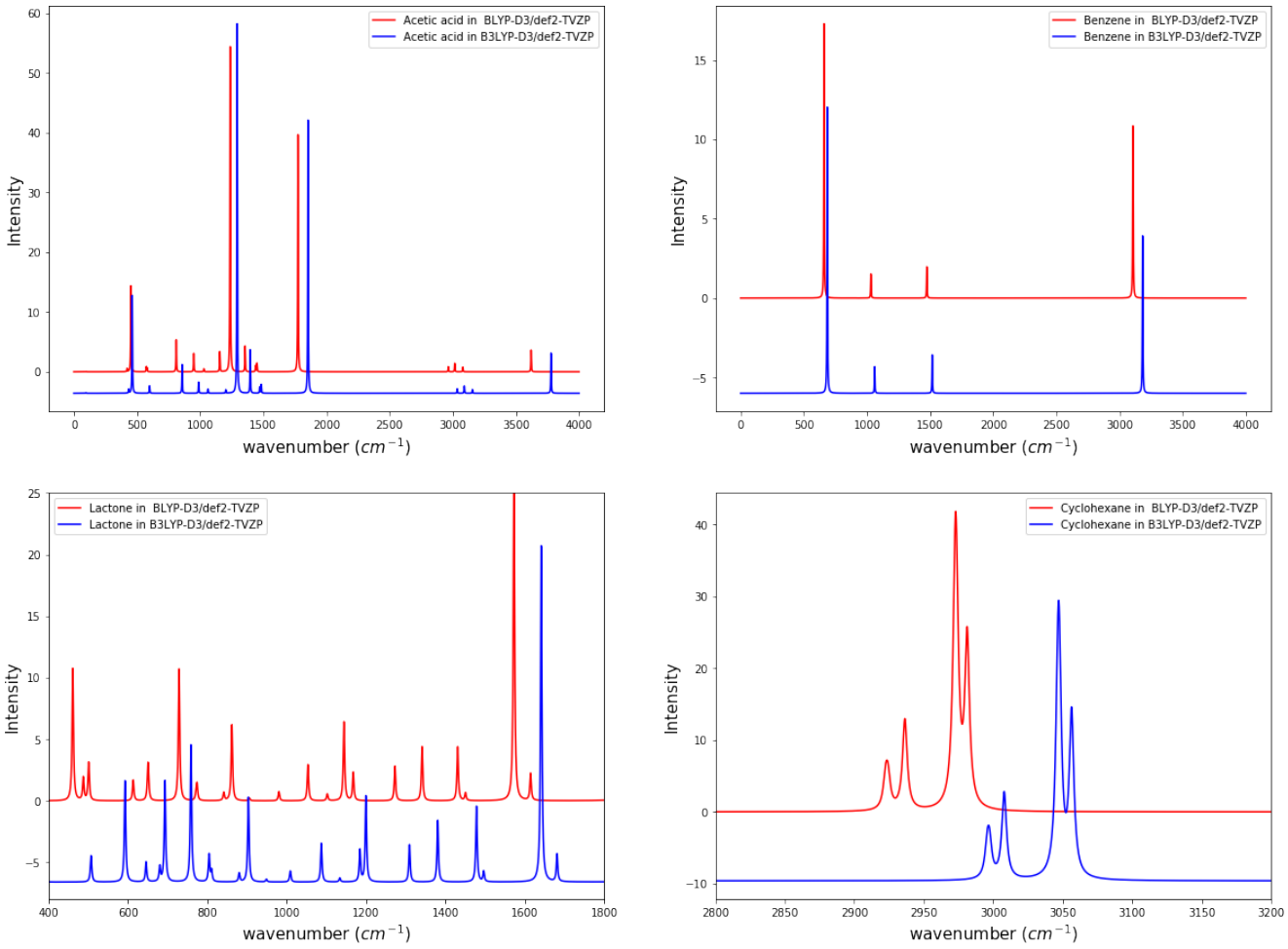
Since we've calculated twice with BLYP and B3LYP successively, we could import two series of .xyz output files of four molecules as the input file to simulate IR spectroscopy, with ORCA codes below. Herein, we don't need to add "opt" again, since we've got the structure optimized. Then, we only exploit B3LYP method to compute IR spectroscopy, which means that any difference of approaching results would only have the origin in the previous computation of energy by BLYP or B3LYP.

```
vi orca_job_3.inp
> ! RKS B3LYP D3 def2-TZVP
>
> ! AnFreq
>
> * xyzfile 0 1 orca_job_1.xyz
|
orca orca_job_3.inp > orca_job_3.out

vi orca_job_4.inp
> ! RKS B3LYP D3 def2-TZVP
>
> ! AnFreq
>
> * xyzfile 0 1 orca_job_2.xyz
|
orca orca_job_4.inp > orca_job_4.out
```

In fact, since we could only acquire separated wavenumbers and related intensities, we seek to fit the data extracted manually from each output file with Gaussian-type functions in *Python*, for the sake of the smooth and continuous plots. Below is the code:

```
def get_IR(data, intensities):
    X = np.linspace(0, 4000, 100000)
    tmp = np.array([grid.map_on_posgrid(np.array([_x]), X[None, :], 2,
      mode='lorentzian', dim=1) for _x in data])
    IR = tmp * intensities[:, None]
|
    return X, IR.sum(axis=0)
```

Finally, we could illustrate IR plots below, comparing the differences obtained from BLYP and B3LYP.



2.2 Analyses and Comparaison

According to the plots above, we could easily see that with the more accurate method, B3LYP would furnish the blue-shift results in all cases, demonstrating that BLYP always furnish lower energy results. Then, we'd like to analyse four cases respectively as below.

- **Benzene**

  Owing to the rather simple and symmetrical skeleton, there's few change for benzene. In the table below, "No." means the order appearing in the output files. And the representative wavenumbers of benzene are listed here, for the sake of comparaison.

No.	Wavenumber (cm^{-1}) BLYP	Intensity (T^2) BLYP	Wavenumber (cm^{-1}) B3LYP	Intensity (T^2) B3LYP
10	660.64	108.612648	686.04	113.318679
19	1031.20	4.808587	1061.19	5.126445
20	1031.74	4.910683	1061.46	5.464994
26	1474.09	6.109583	1516.84	7.287464
27	1474.41	6.351062	1517.25	8.068488
33	3105.82	42.582380	3182.39	35.386656
34	3107.87	43.458039	3183.98	36.803752

- **Cyclohexane**

  As for cyclohexane, due to the same raison, there's also few change shown in IR spectroscopy. And the representative wavenumbers are listed:

No.	Wavenumber (cm^{-1}) BLYP	Intensity (T^2) BLYP	Wavenumber (cm^{-1}) B3LYP	Intensity (T^2) B3LYP
42	2922.53	25.239643	2995.78	26.304329
44	2924.29	25.832215	2997.36	26.591240
47	2936.41	79.301988	3007.90	75.617496
48	2972.58	134.508324	3046.70	126.166028
49	2973.38	130.049457	3047.51	122.499699
53	2981.23	147.049019	3056.41	140.908559

- **Acetic acid**

  However, for the acetic acid, there are some alterations for the intensity, emphasized as the bold below. Experimentally, there would be a swelling shown in IR of acetic acid, but we could see here.

No.	Wavenumber (cm^{-1}) BLYP	Intensity (T^2) BLYP	Wavenumber (cm^{-1}) B3LYP	Intensity (T^2) B3LYP
8	451.18	90.265058	462.10	102.864657
11	810.64	33.519336	857.16	30.131401
15	1237.65	341.930603	1292.68	**388.561758**
16	1355.61	26.968759	1396.25	45.643198
19	1774.61	249.394409	1854.14	**287.260248**
23	3620.60	22.798584	3778.72	**42.193699**

- **Lactone**

  Similarly, the lactone would give out some differences for the intensity. In Moreover, it's quite interesting that under two calculations, the same vibration mode would appear in the different order. For example, 650.21 from BLYP has the same vibration mode with 693.16 from B3LYP:

No.	Wavenumber (cm^{-1}) BLYP	Intensity (T^2) BLYP	Wavenumber (cm^{-1}) B3LYP	Intensity (T^2) B3LYP
14	650.21	14.877459	680.07	7.482871
15	651.50	6.284061	693.16	51.579908
17	771.29	2.842172	804.31	13.919922
18	773.83	8.175797	810.61	5.494373
26	1144.16	40.207221	1184.37	16.168763
27	1167.52	14.378592	1199.48	43.809425

3 MO Energies and Figures

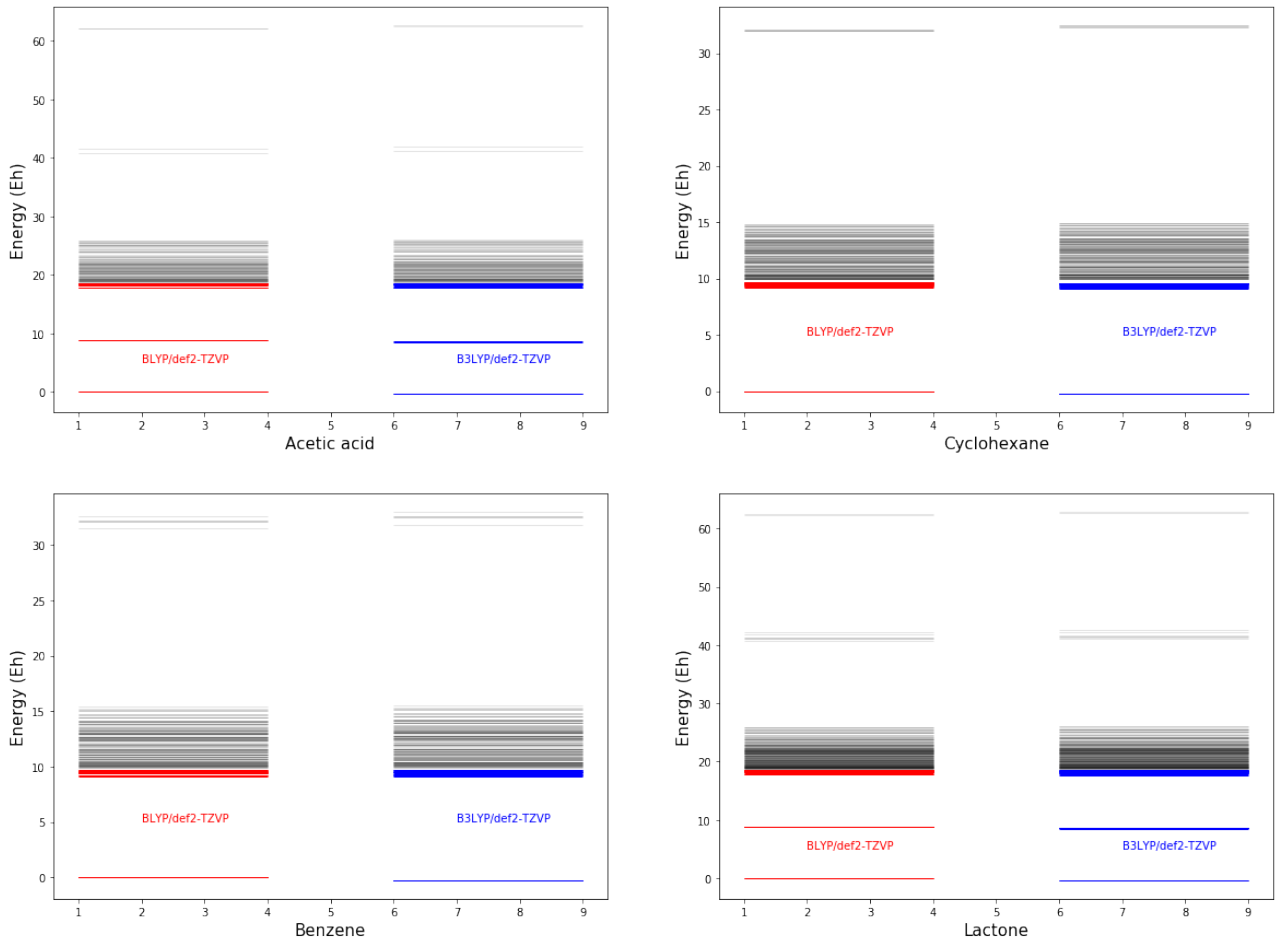
In chemistry, *Molecular Orbital* (MO) is a mathematical function describing the wave-like behavior of an electron in a molecule. This function could be used to calculate chemical and physical properties such as the probability of finding an electron in any specific region. The term orbital was introduced by Robert S. Mulliken in 1932 as an abbreviation for one-electron orbital wavefunction.[20] Molecular orbitals are usually constructed by combining atomic orbitals or hybrid orbitals from each atom of the molecule, or other molecular orbitals from groups of atoms.

$$\Psi = \sum_i c_i \phi_i$$

In chemistry, HOMO and LUMO are types of molecular orbitals. These acronyms stand for *Highest Occupied Molecular Orbital* and *Lowest Unoccupied Molecular Orbital*, respectively. Both are addressed as *Frontier Molecular Orbital* (FMO). [21] Herein, we'd like to take a look at the distributions of each molecule energy, and then some typical figures as well.

3.1 HOMO and LUMO

Firstly, we find out all related orbital energies from output files from IR calculations, then import them inside *Python*. Suppose that the minimum energy from BLYP/def2-TZVP is equal to 0, we subtract this minimum energy from energies of all different levels. Occupied orbitals of BLYP are printed in red, while those of B3LYP in blue; yet all unoccupied orbitals are printed in grey. Below are the overview for energies for each molecule.



Additionally, we'd like to expand these plots to figure out positions of HOMO and LUMO for all molecules. Actually, in the basis set of def2-TZVP, one carbon atom would contain 5, 3, 2, 1 orbitals for s, p, d, f respectively.

Taking angular momentum into account, the total number of orbitals would be:

$$N_{\mathrm{C}} = 5 \times 1 + 3 \times 3 + 2 \times 5 + 1 \times 7 = 31$$

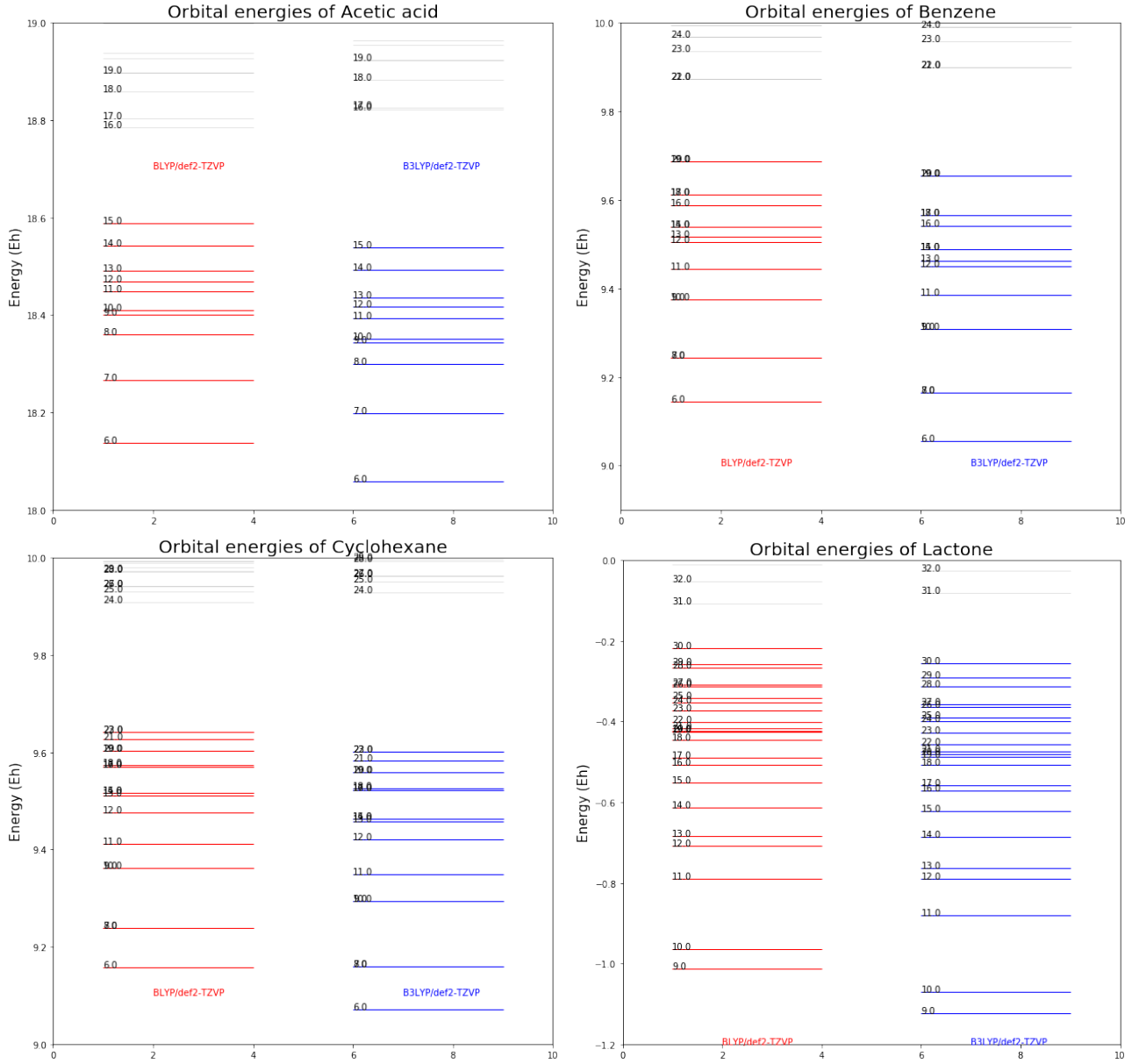
As for the hydrogen atom, there would be six orbitals. Therefore, as for benzene, the total number of MO is just

$$N_{\mathrm{MO-benzene}} = (31 + 6) \times 6 = 222$$

Also, the total number of MO for cyclohexane would be

$$N_{\mathrm{MO-cyclohexane}} = 31 \times 6 + 6 \times 12 = 258$$

According to FMO theory, it attach great importance to follow the transfer of electrons among FMOs during a chemical reaction. Thus we label all orbitals with the help of Python in order and then expand the view, for the sake of checking their exact positions much clearly. Since there would be degenerated orbitals with close energies, some numbers overlap as below.



3.2 Visualisation of MO

According to the figures above, we could trace the number of HOMO and LUMO orbitals for each molecule.

Molecule	HOMO-1	HOMO	LUMO	LUMO+1
Acetic acid	14	15	16	17
Benzene	19	20	21	22
Cyclohexane	22	23	24	25
Lactone	29	30	31	32

In order to visualize these orbitals, we utilize ORCA again, with the command "orca_plot" in this case. In general, we could type "orca_plot gbw-file -I" to enter the interaction page. Here, we introduce the template file as below, including all necessary parameters.

```
1      # PlotType - Type of plot to be done
7      # Format   - File format for output file
XXX    0 # MO and operator (if not density)
0      # State density to be plotted
-
MO_XXX.cube      # Output file
200    # ncont    - number of contours
1      # icont    - contour option
0      # Skeleton - flag for Skeleton plotting
0      # Atoms    - flag for atom plotting
0      # UseCol   - flag for use of color
100  100  100                          # Grid dimensions
-20.000000     20.000000                # X dimension
-20.000000     20.000000                # Y dimension
-20.000000     20.000000                # Z dimension
-1   -1   -1                           # defining atoms
0.000000      0.000000      0.000000 # defining vector-1
1.000000      0.000000      0.000000 # defining vector-2
0.000000      1.000000      0.000000 # defining vector-3
```

And then we could execute loops inside Terminal directly to avoid input parameters manually sequently, getting a series of .cube files

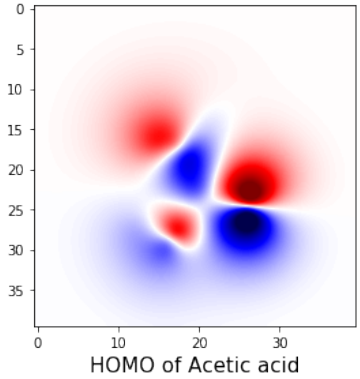
```
#!/bin/bash

for i in {15..25}; do
  echo $i
  sed 's/XXX/'$i'/g' plot_template.asc.txt > tmp
  orca_plot orca_job_3.gbw tmp > orca_plot.log
  rm -f tmp
done
```

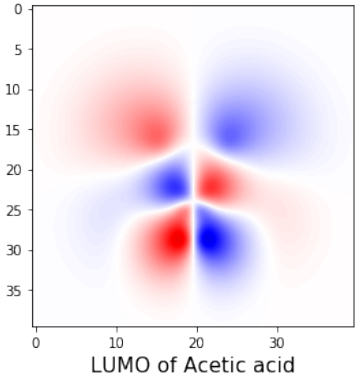
Then, we could open these .cube files with Avogadro.app, exploring the shape of each MO with the command "Extensions-Create Surfaces-Surface".

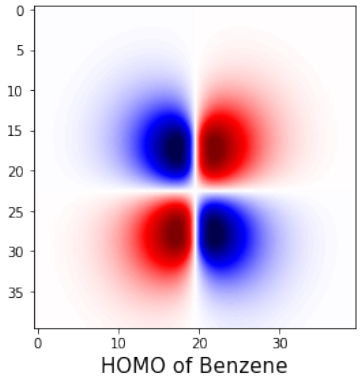
Alternatively, we could continue executing *Python* programmes, extracting coordinates from .cube files and projecting them along one specific axis, with the code like:

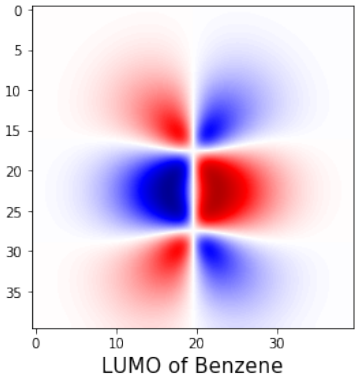
```
qq1H = quantum.WaveFunction('.../MO_15.cube')
data_1H = qq1H.data.sum(axis=2)
ax[0,0].imshow(data_1H,
           interpolation='bicubic',
           cmap=cm.seismic,
           norm=colors.Normalize(vmin=-1.5,vmax=+1.5))
```

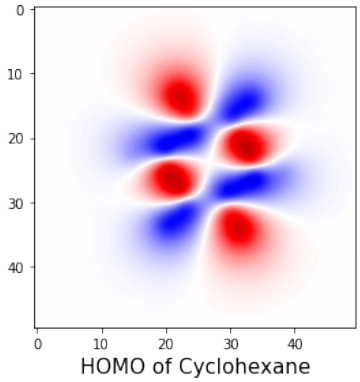
Since benzene is much symmetric and phases of orbitals would cancel each other out, there would no projection result from some directions. Below are plots of all four molecule orbitals.

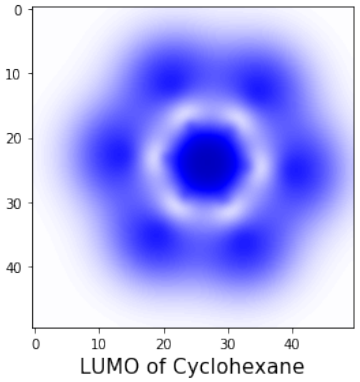


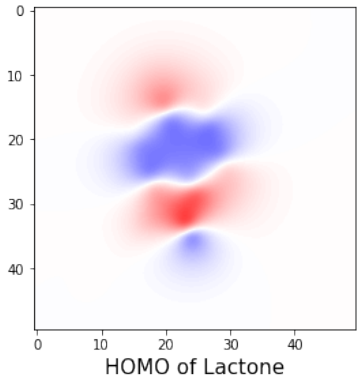


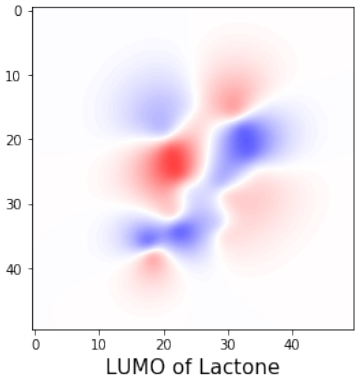












4 Machine Learning

With the advent of an unprecedented era teemed with high technologies, scientists gradually utilise various tools to simplify the research. As an excellent representative of them, developed from last century but shining quite increasingly in recent years, *Machine learning* (ML) is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task.[22] Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or infeasible to develop a conventional algorithm for effectively performing the task.

Machine learning is closely related to computational statistics, which focuses on making predictions using computers. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a field of study within machine learning, and focuses on exploratory data analysis through unsupervised learning.[23] In its application across business problems, machine learning is also referred to as predictive analytics.

4.1 Linear Polynomial Regression

To some extent, it seems that the linear regression should not be considered as one machine learning method. However, it's the basis of all current approaches.

In statistics, linear regression is a linear approach to modeling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called multiple linear regression.[24] This term is distinct from multivariate linear regression, where multiple correlated dependent variables are predicted, rather than a single scalar variable.[25]

In linear regression, the relationships are modeled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called linear models.[26] Linear regression was the first type of regression analysis to be studied rigorously, and to be used extensively in practical applications.[27] This is because models which depend linearly on their unknown parameters are easier to fit than models which are non-linearly related to their parameters and because the statistical properties of the resulting estimators are easier to determine.

Linear regression has many practical uses. Most applications fall into one of the following two broad categories:

- If the goal is prediction, forecasting, or error reduction, linear regression can be used to fit a predictive model to an observed data set of values of the response and explanatory variables. After developing such a model, if additional values of the explanatory variables are collected without an accompanying response value, the fitted model can be used to make a prediction of the response.

- If the goal is to explain variation in the response variable that can be attributed to variation in the explanatory variables, linear regression analysis can be applied to quantify the strength of the relationship between the response and the explanatory variables, and in particular to determine whether some explanatory variables may have no linear relationship with the response at all, or to identify which subsets of explanatory variables may contain redundant information about the response.

Linear regression models are often fitted using the least squares approach, but they may also be fitted in other ways, such as by minimizing the "lack of fit" in some other norm (as with least absolute deviations regression), or by minimizing a penalized version of the least squares cost function as in Ridge regression (L^2-norm penalty) and Lasso (L^1-norm penalty). Conversely, the least squares approach can be used to fit models that are not linear models. Thus, although the terms "least squares" and "linear model" are closely linked, they are not synonymous.

Given a data set $\{y_i, x_{i1}, \cdots, x_{ip}\}_{i=1}^{n}$ of n statistical units, a linear regression model assumes that the relationship between the dependent variable y and the p-vector of regressors x is linear. This relationship is modeled through a disturbance term or error variable ϵ — an unobserved random variable that adds "noise" to the linear relationship between the dependent variable and regressors. Thus the model takes the form:

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \epsilon_i = \mathbf{x_i^T}\boldsymbol{\beta} + \epsilon_i$$

Or written in matrix notation as:

$$\mathbf{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

where $\mathbf{y}$, $\boldsymbol{\beta}$, $\boldsymbol{\epsilon}$ are vectors and $\boldsymbol{X}$ is a matrix. With data of $\boldsymbol{X}$ and $\mathbf{y}$, we could fit the unknown parameters β and ϵ; while with known parameters β and ϵ, we could predict y_i from $\mathbf{x_i^T}$.

The Lennard-Jones potential (also termed the L-J potential, 6-12 potential, or 12-6 potential) is a mathematically simple model that approximates the interaction between a pair of neutral atoms or molecules, first proposed in 1924 by John Lennard-Jones. [28] The most common expressions of the L-J potential are

$$V_{\mathrm{LJ}} = 4\epsilon[(\frac{\sigma}{r})^{12} - (\frac{\sigma}{r})^{6}] = \epsilon[(\frac{r_m}{r})^{12} - (\frac{r_m}{r})^{6}]$$

$$r_{\mathrm{m}} = 2^{1/6}\sigma \approx 1.122\sigma$$

where ϵ is the depth of the potential well, σ is the finite distance at which the inter-particle potential is zero, r is the distance between the particles, and r_m is the distance at which the potential reaches its minimum. At r_m, the potential function has the value ϵ.

Next, with the available data in hand, we'd like to use linear regression to interpolate *Hydration Free Energies* (HFE) of charged Lennard-Jones spheres. Suppose that HFE is one function of q, σ, and ϵ, the simplest function would be:

$$f(q, \sigma, \epsilon) = aq + b\sigma + c\epsilon$$

We could follow the processes above to figure out the value of each parameter. In order to fit better, we could increase the polynomial degree to 2 such as:

$$f(q, \sigma, \epsilon) = aq + b\sigma + c\epsilon + dq^2 + eq\sigma + fq\epsilon + g\sigma^2 + h\sigma\epsilon + i\epsilon^2$$

Right now the task is fitting the whole data and seeking values of coefficients. And in general cases, it should be written as below, where c_{ijk} is the exact constant for each item, waiting for the calculation. In the way, the degree n would be the only parameter here.

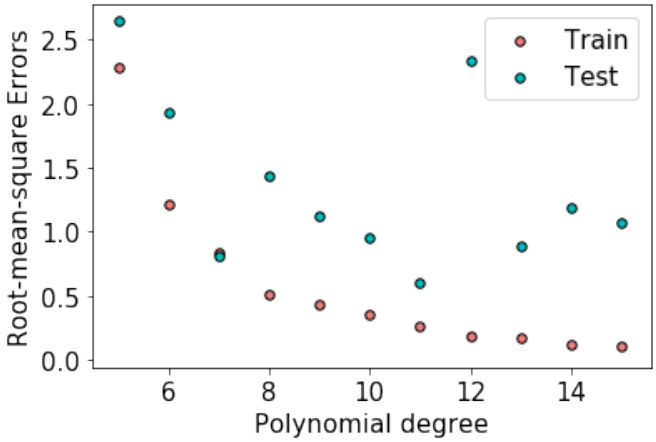
$$f(q, \sigma, \epsilon) = \sum_{0<i+j+k\leqslant n} c_{ijk} q^i \sigma^j \epsilon^k$$

Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. [29] With such a package, we would divide all data into two parts, as the training and the test respectively, and then evaluate the fitting calculation. Below are related codes.

```
from sklearn.preprocessing import PolynomialFeatures
degree = 2
poly = PolynomialFeatures(degree,include_bias=False)
X_poly = poly.fit_transform(X)
X_poly_feature_name = poly.get_feature_names(['q','sig','eps'])
print('New features (%d) : %s' % (len(X_poly_feature_name),X_poly_feature_name))
# Save new features into a pandas dataframe
df_poly = pd.DataFrame(X_poly, columns=X_poly_feature_name)
df_poly['HFE']=df['HFE']
df_poly = df_poly.dropna()
df_poly.head()
#
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
        train_test_split(df_poly.drop('HFE',axis=1), df['HFE'],test_size=0.1)
linear_model = LinearRegression(normalize=True)
model_poly=linear_model.fit(X_train,y_train)
y_poly_train = model_poly.predict(X_train)
RMSE_poly=sqrt(mean_squared_error(y_poly_train,y_train))
```

```
print("Root-mean-square error for the training set of simple polynomial model:",RMSE_poly)
y_poly_test = model_poly.predict(X_test)
RMSE_poly=sqrt(mean_squared_error(y_poly_test,y_test))
print("Root-mean-square error for the test set of simple polynomial model:",RMSE_poly)
```

Fitting the data under different polynomial degrees, we could obtain the figure as the degree varies from 5 to 15. Indeed, in mathematics and its applications, the root mean square (RMS or rms) is defined as the square root of the mean square (the arithmetic mean of the squares of a set of numbers): $x_{\mathrm{RMS}} = \sqrt{(\sum_{i=1}^{n} x_i^2)/n}$



Obviously, the errors of training data converge constantly; however, the errors of test data just diverge. According to the figure above, the optimal degree should be 7, considering the time consumed. Actually, with lower or higher degrees, errors would be very large, in which we would describe as underfitting and overfitting. It would be simple to add variables to solve the underfitting; however, as for avoiding the overfitting, one possible method is called "regularization".

4.2 LASSO Regularization

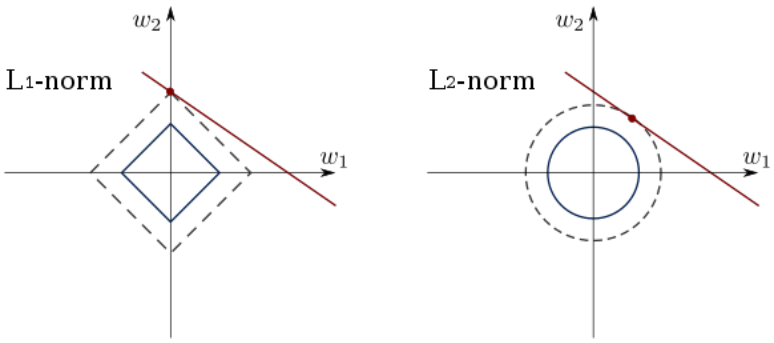
Least Absolute Shrinkage and Selection Operator, namely LASSO, is one approach for regression analysis, firstly introduced in geophysics literature in 1986 [30] and then revealed in 1996. [31]

Lasso was originally formulated for least squares models and this simple case reveals a substantial amount about the behavior of the estimator, including its relationship to ridge regression and best subset selection and the connections between lasso coefficient estimates and so-called soft thresholding. It also reveals that (like standard linear regression) the coefficient estimates do not need to be unique if covariates are collinear. Though originally defined for least squares, lasso regularization is easily extended to a wide variety of statistical models including generalized linear models, generalized estimating equations, proportional hazards models, and M-estimators, in a straightforward fashion. [32]

Lasso was originally introduced in the context of least squares, and it can be instructive to consider this case first, since it illustrates many of lasso's properties in a straightforward setting. Consider a sample consisting of N cases, each of which consists of p covariates and a single outcome. Let y_i be the outcome and $x_i = (x_1, x_2, \ldots, x_p)^{\mathrm{T}}$ be the covariate vector for the i^{th} case. Then the objective of lasso is to solve:

$$\min_{\beta_0,\beta} \sum_{i=1}^{N} (y_i - \beta_0 - x_i^T \beta)^2 \text{ subject to } \sum_{j=1}^{p} |\beta_j| \leqslant t$$

Since the order of β_j is 1, Lasso is also called L^1-norm penalty method; while Ridge regression contains L^2-norm penalty shown below.

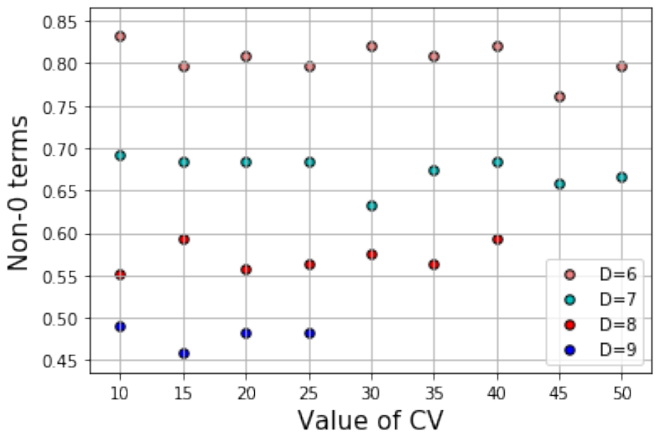


However, we have to set the original penalty parameter manually at first. And in some cases we could not furnish an excellent value, especially confronting with some unknown situations. Therefore, we introduce one modified version of Lasso, namely LassoCV, which focuses on cross-validation by the iterating fitting based on the traditional Lasso method. [33] During the cross-validation, the test data would be different in each iteration, as the training data is the rest part inside the whole data, which is the core of regularization. Herein, we would concentrate on two parameters, namely "degree" and "CVs", which refer to the order of polynomial, as well as the number of folds as the cross-validation splitting strategy. With codes below, we would vary these two parameters mainly, figuring out the optimal polynomial degree and following the variation of errors like RMS.

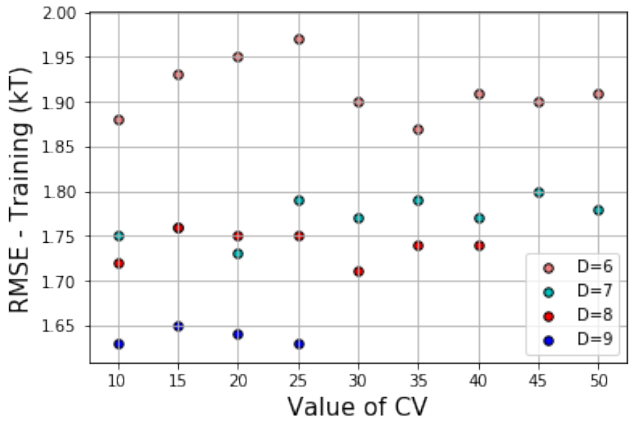
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LassoCV
from sklearn.pipeline import make_pipeline
X_train, X_test, y_train, y_test =
        train_test_split(df.drop('HFE',axis=1), df['HFE'],test_size=0.1)

# Alpha (regularization strength) of LASSO regression
lasso_eps = 1e-5
lasso_nalpha = 100
lasso_iter = 250000
CVs = 10
degree = 6
model = make_pipeline(PolynomialFeatures(degree, interaction_only=False),
                      LassoCV(eps=lasso_eps,n_alphas=lasso_nalpha,max_iter=lasso_iter,
                              normalize=True,cv=CVs))
model.fit(X_train,y_train)

# Statistical measures
train_pred = model.predict(X_train)
RMSE_train = sqrt(mean_squared_error(train_pred,y_train))
MAE_train = mean_absolute_error(train_pred,y_train)
R_train = model.score(X_train,y_train)
print('Deg : \%d \t RMSE train : \%.2f kT \t MAE train : \%.2f kT \t R^2 train : \%.2f'
      \% (degree,RMSE_train,MAE_train,R_train))
test_pred = model.predict(X_test)
RMSE_test = sqrt(mean_squared_error(test_pred,y_test))
MAE_test = mean_absolute_error(test_pred,y_test)
R_test = model.score(X_test,y_test)
print('Deg : \%d \t RMSE test : \%.2f kT \t MAE test : \%.2f kT \t R^2 test : \%.2f'
      \% (degree,RMSE_test,MAE_test,R_test))

# Check how many fetures are put to zero
coeff1 = model[-1].coef_
nbterm = len(coeff1)
```

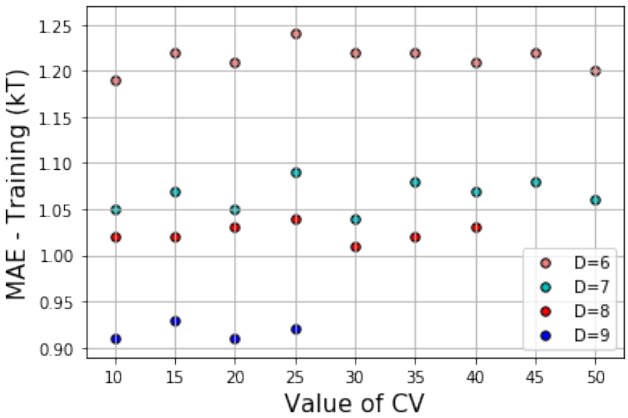
```
nbterm_no0 = len(coeff1[coeff1!=0])
print('Deg : \%d \t non-0 terms : \%d/\%d ' \% (degree, nbterm_no0, nbterm))
```

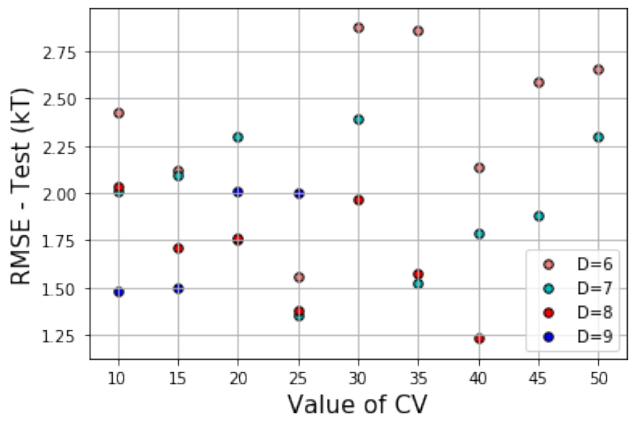
Below is the plot for non-zero terms with the increase of CV.

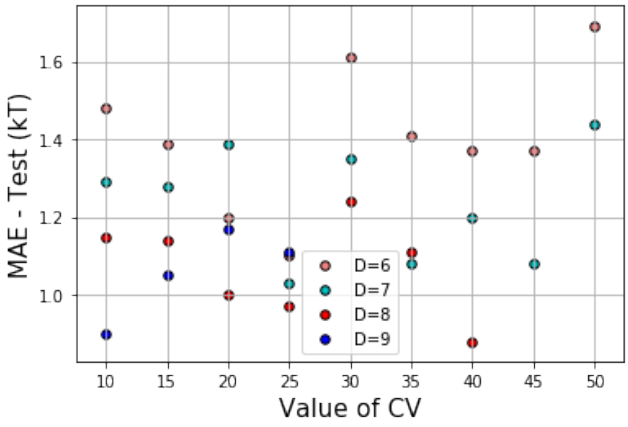


Obviously, by adding the number of degree, the percentage of non-zero coefficients would decrease. At the same time, the value of CV does not quite matter to some extent. Notice that the higher degree, the longer time for the calculation. Then, there are figures for errors, where RMSE means root mean square error and MAE means mean absolute error. $x_{\text{MAE}} = (\sum_{i=1}^{n} |x_i - \hat{x}_i|)/n$







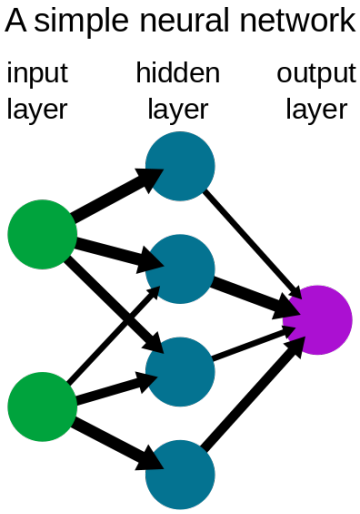


On the one hand, as for training data, both RMSE and MAE would walk around in a certain range for each degree. On the other hand, these two errors would diverge in some degree as for test data. In summary, the optimal degree would be 8 and the optimal CV could be 25 according to results above.

4.3 Neural Network Model

A neural network is a network or circuit of neurons, or in a modern sense, an artificial neural network, composed of artificial neurons or nodes. [34] Thus a neural network is either a biological neural network, made up of real biological neurons, or an artificial neural network, for solving artificial intelligence (AI) problems. The connections of the biological neuron are modeled as weights. A positive weight reflects an excitatory connection, while negative values mean inhibitory connections. All inputs are modified by a weight and summed. This activity is referred as a linear combination. Finally, an activation function controls the amplitude of the output. For example, an acceptable range of output is usually between 0 and 1, or it could be -1 and 1.

These artificial networks might be used for prediction, adaptive control and applications where they can be trained via a dataset. Self-learning resulting from experience can occur within networks, which can derive conclusions from a complex and seemingly unrelated set of information. [35] Below is a simple model of neural network.



In this part, we'd like make us of unsupervised one-layer neural network model to fit the polynomial coefficients, seeking the optimal degree moreover. Actually, the number of terms under one certain degree would just be the number of input nodes. Empirically, we select the number of hidden nodes is the average of the sum between the number of input nodes and that of output nodes, which equals to 1 in this case as HFE.

Even though the programme could give out the number of input nodes directly later, we could still compute it easily. As for the term $q^i\sigma^j\epsilon^k$, we could fix i at first and then account how much terms with the condition $j+k \leqslant d-i$ there would be; simply, it should be $(n-i)(n-i+1)/2$. Then, changing the value of i from 0 until the degree, we add all the numbers above. We know the sum of squares:

$$\sum_{a=1}^{n} a^2 = n(n+1)(2n+1)/6$$

Finally, subtracting the specific term as $i=j=k=0$, we get the number below:

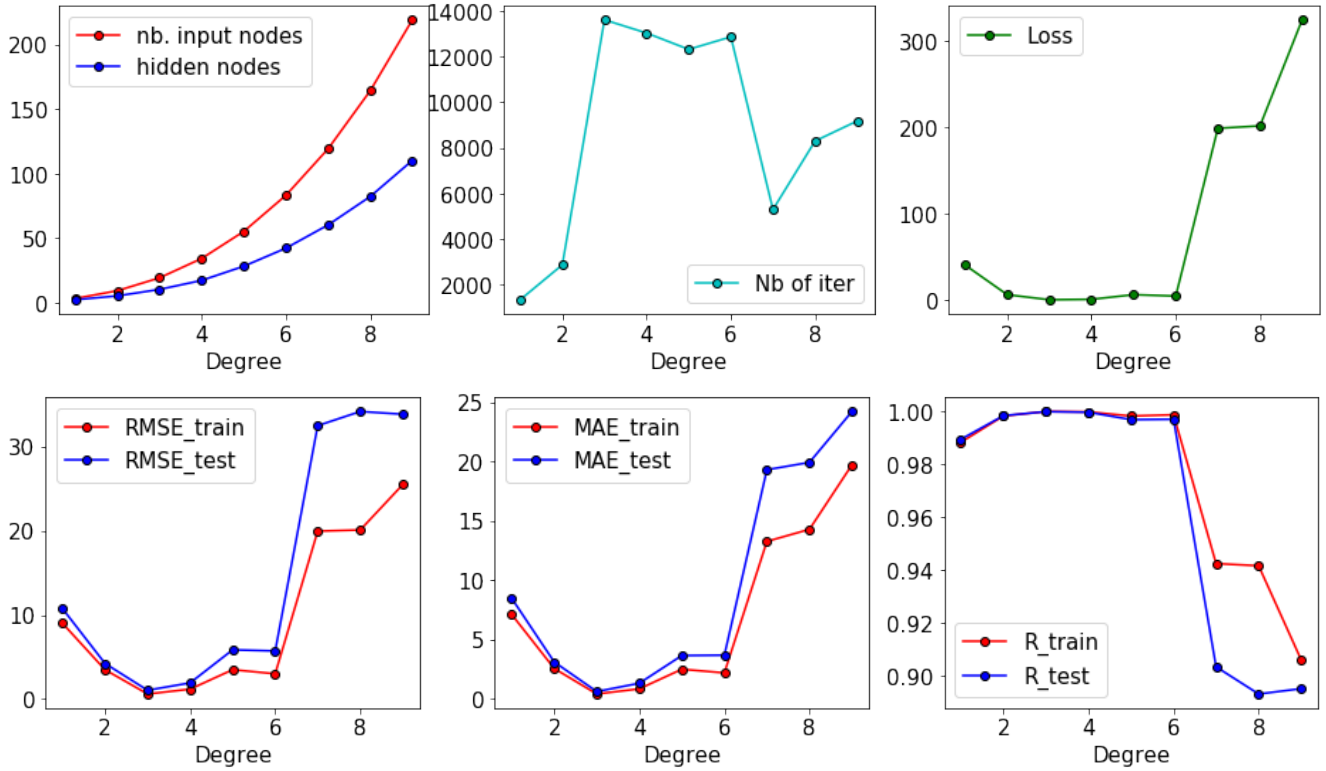
$$N_{\text{terms}} = (\sum_{i=1}^{d} \frac{(d-i)(d-i+1)}{2}) - 1 = (\sum_{i=1}^{d} \frac{i(i+1)}{2}) - 1 = \frac{d(d+1)(d+2)}{6} - 1$$

Below is the code for the neural network:

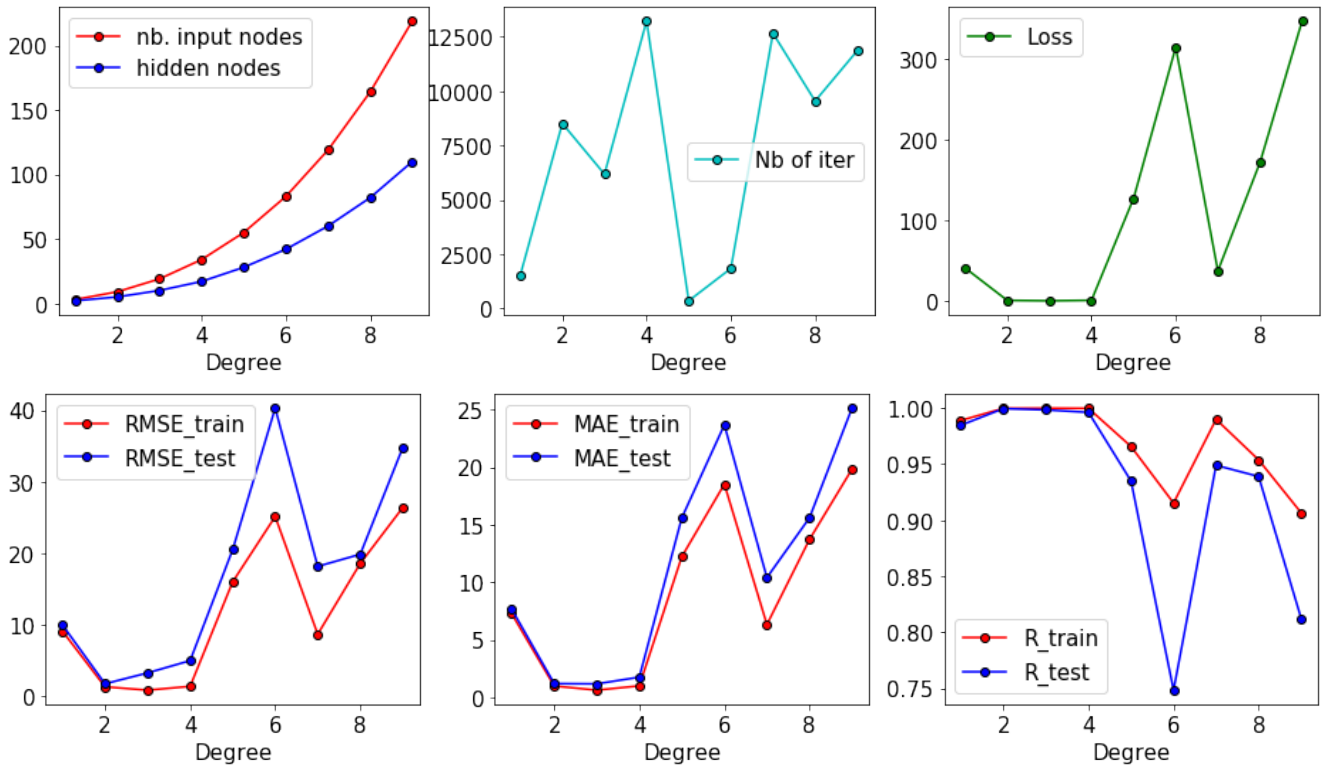
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.neural_network import MLPRegressor
from sklearn.pipeline import make_pipeline
from scipy.special import comb
degree_min = 1
degree_max = 9
```

```
| for degree in range(degree_min,degree_max+1):
|     ninput = comb(degree+3,3) - 1
|     hidden = int((ninput+1)/2)

| #assignment
|     x_input_node[degree]=ninput
|     x_hidden_node[degree]=hidden
|     print('Polynomial degree : \%d' \% (degree))
|     print('nb. input nodes : \%d' \% (ninput))
|     print('nb. hidden nodes in one layer : \%d' \% (hidden))
|     model = make_pipeline(PolynomialFeatures(degree, interaction_only=False),
|                     MLPRegressor(solver='lbfgs', alpha=1e-4, hidden_layer_sizes=(10,10),
|                                  max_iter=100000,random_state=1,activation='relu',
|                                  batch_size=batchsize,tol=1e-5,early_stopping=True))
|     model.fit(X_train,y_train)

| #assignment
|     x_iteration[degree]=model[-1].n_iter_
|     x_los[degree]=model[-1].loss_
|     print('Nb of iter : \%d\t Loss : \%.2f' \% (model[-1].n_iter_,model[-1].loss_))
|     train_pred = model.predict(X_train)
|     RMSE_train = sqrt(mean_squared_error(train_pred,y_train))
|     MAE_train = mean_absolute_error(train_pred,y_train)
|     R_train = model.score(X_train,y_train)

| #assignment
|     x_RMSE_train[degree]=RMSE_train
|     x_MAE_train[degree]=MAE_train
|     x_R_train[degree]=R_train
|     print('Deg : \%d\t RMSE train : \%.2f kT\t MAE train : \%.2f kT\t R^2 train : \%.2f'
|           \% (degree,RMSE_train,MAE_train,R_train))
|     test_pred = model.predict(X_test)
|     RMSE_test = sqrt(mean_squared_error(test_pred,y_test))
|     MAE_test = mean_absolute_error(test_pred,y_test)
|     R_test = model.score(X_test,y_test)

| #assignment
|     x_RMSE_test[degree]=RMSE_test
|     x_MAE_test[degree]=MAE_test
|     x_R_test[degree]=R_test
|     print('Deg : \%d\t RMSE test : \%.2f kT\t MAE test : \%.2f kT\t R^2 test : \%.2f'
          \% (degree,RMSE_test,MAE_test,R_test))
```

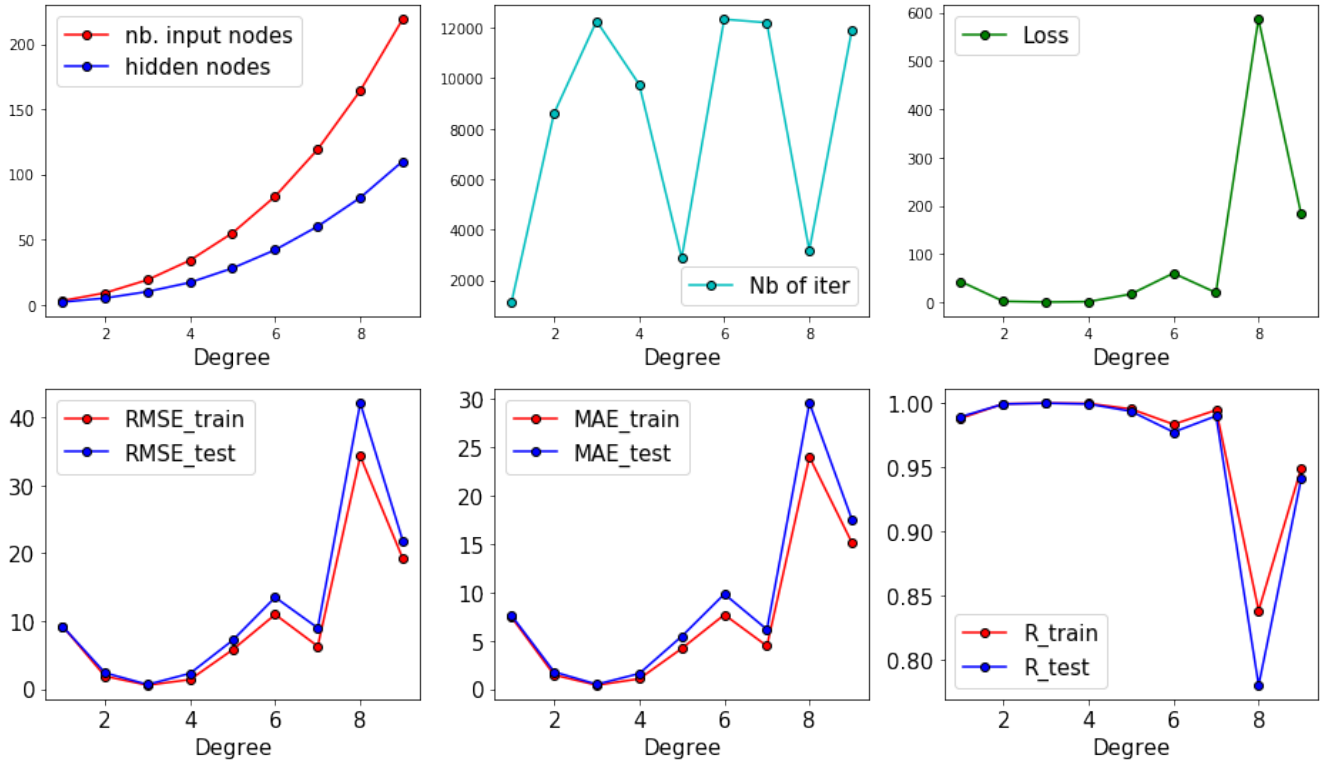
With a series of results in hand, containing numbers of nodes and iteration, RMSE, MAE and correlation coefficient, we then draw these plots as below:



Herein, it's quite interesting that since there are random factors inside the algorithm, different results would be furnished by the separated launch of the programme. For example, here is another attempt:



Also, here is the third attempt:



Unlike the polynomial model, neural network would enter the overfitting regime much earlier with the much lower degree. Here, when degree is equal to 3, there would be the optimal result, with the lowest errors.

4.4 Brief Summary

As a function of q, σ, and ϵ, HFE could be written as a polynomial based on the charged Lennard-Jones spheres.

$$f(q,\sigma,\epsilon) = \sum_{0<i+j+k\leqslant n} c_{ijk} q^i \sigma^j \epsilon^k$$

In the section, we fit the correspondant coefficients by three approaches, including linear regression, LASSO regularization, and neural network. Apparently, to avoid the underfitting and overfitting, we'd better choose one optimal degree of the polynomial. However, for the neural network, the optimal degree would be much lower than that obtained from the other two ways, revealing the fundamental difference among them.

Methods	Linear regression	Lasso regularization	Neural Network
Optimal degree	7	8	3
RMSE - train	0.84	1.75	0.59
RMSE - test	0.81	1.38	1.03
MAE - train		1.04	0.43
MAE - test		0.97	0.63

Acknowledgements

The program, Discovery week in the laboratory of Chemistry Department of ENS, especially the Theoretical Group, is highly acknowledged. Thanks Prof. Marie Labeye for the notification. Thanks Prof. Rodolphe Vuilleumier for the laboratory. Thanks Dr. Sascha Jähnigen so much for his exhaustive instructions around a series of calculations and simulations. Thanks Ms. Sohvi Luukkonen so much for her kind help toward the machine learning part, especially test data and programmes.

References

[1] Schrödinger, E. "An Undulatory Theory of the Mechanics of Atoms and Molecules". Phys. Rev. 1926, 28(6): 1049–1070.

[2] M. Born; R. Oppenheimer. Zur Quantentheorie der Molekeln. Annalen der Physik. 1927, 389(20): 457–484

[3] Hartree, D. R. "The Wave Mechanics of an Atom with a Non-Coulomb Central Field". Math. Proc. Camb. Philos. Soc. 1928, 24(1): 111.

[4] Fock, V. A. "Näherungsmethode zur Lösung des quantenmechanischen Mehrkörperproblems". Z. Phys. 1930, 61(1): 126.

[5] Fock, V. A. "Selfconsistent field" mit Austausch für Natrium". Z. Phys. 1930, 62(11): 795.

[6] Slater, J. C. "The Self Consistent Field and the Structure of Atoms". Phys. Rev. 1928, 32 (3): 339.

[7] Čížek, Jiří. "On the Correlation Problem in Atomic and Molecular Systems. Calculation of Wavefunction Components in Ursell-Type Expansion Using Quantum-Field Theoretical Methods". The Journal of Chemical Physics. 1966, 45(11): 4256–4266.

[8] Møller, Christian; Plesset, Milton S. "Note on an Approximation Treatment for Many-Electron Systems". Phys. Rev. 1934, 46(7): 618–622.

[9] A. D. Becke, Physical Review A 1988, 38(6): 3098–3100.

[10] Lee, C.; Yang, W.; Parr, R. G. Development of the Colle-Salvetti correlation-energy formula into a functional of the electron density. Physical review B, 1988, 37(2): 785.

[11] Miehlich, B.; Savin, A.; Stoll, H., Preuss, H. Results obtained with the correlation energy density functionals of Becke and Lee, Yang and Parr. Chemical Physics Letters, 1989, 157(3): 200-206.

[12] A. D. Becke, Journal of Chemical Physics. 1993, 98(7): 5648–5652.

[13] P. J. Stephens, F. J. Devlin, C.F . Chabalowski and M. J. Frisch, Journal of Physical Chemistry 1994, 98(45): 11623–11627.

[14] A. Schafer, H. Hornand R. Ahlrichs, Journal of Chemical Physics 1992, 97(4): 2571–2577.

[15] A. Schafer, C. Huberand R. Ahlrichs, Journal of Chemical Physics 1994, 100(8): 5829–5835.

[16] Neese, Frank. "The ORCA program system". Wiley Interdisciplinary Reviews: Computational Molecular Science. 2012, 2(1): 73–78.

[17] Neese, Frank (2018). "Software update: The ORCA program system, version 4.0". Wiley Interdisciplinary Reviews: Computational Molecular Science. 2018, 8(1): e1327.

[18] Hanwell, Marcus D; Curtis, Donald E; Lonie, David C; Vandermeersch, Tim; Zurek, Eva; Hutchison, Geoffrey R. "Avogadro: An advanced semantic chemical editor, visualization, and analysis platform". J. Cheminform. 2012, 4(1): 17.

[19] Michael Rowan-Robinson (2013). Night Vision: Exploring the Infrared Universe. Cambridge University Press. p. 23.

[20] Mulliken, Robert S. "Electronic Structures of Polyatomic Molecules and Valence. II. General Considerations". Phys. Rev. 1932, 41(1): 49–71.

[21] Fukui, Kenichi; Yonezawa, Teijiro; Shingu, Haruo. A Molecular Orbital Theory of Reactivity in Aromatic Hydrocarbons. The Journal of Chemical Physics. 1952, 20(4): 722.

[22] Bishop, C. M. (2006), Pattern Recognition and Machine Learning, Springer, ISBN 978-0-387-31073-2

[23] Friedman, Jerome H. "Data Mining and Statistics: What's the connection?". Computing Science and Statistics. 1998, 29(1): 3–9.

[24] David A. Freedman (2009). Statistical Models: Theory and Practice. Cambridge University Press. p. 26.

[25] Rencher, Alvin C.; Christensen, William F. (2012), "Chapter 10, Multivariate regression – Section 10.1, Introduction", Methods of Multivariate Analysis, Wiley Series in Probability and Statistics, 709 (3rd ed.), John Wiley & Sons, p. 19, ISBN 9781118391679.

[26] Hilary L. Seal. "The historical development of the Gauss linear model". Biometrika. 1967, 54(1/2): 1–24.

[27] Yan, Xin (2009), Linear Regression Analysis: Theory and Computing, World Scientific, pp. 1–2.

[28] Lennard-Jones, J. E. "On the Determination of Molecular Fields", Proc. R. Soc. Lond. A, 1924, 106(738): 463–477.

[29] Fabian Pedregosa; Gaël Varoquaux; Alexandre Gramfort; Vincent Michel; Bertrand Thirion; Olivier Grisel; Mathieu Blondel; Peter Prettenhofer; Ron Weiss; Vincent Dubourg; Jake Vanderplas; Alexandre Passos; David Cournapeau; Matthieu Perrot; Édouard Duchesnay. "Scikit-learn: Machine Learning in Python". Journal of Machine Learning Research. 2011, 12: 2825–2830.

[30] Santosa, Fadil; Symes, William W. "Linear inversion of band-limited reflection seismograms". SIAM Journal on Scientific and Statistical Computing. SIAM. 1986, 7(4): 1307–1330.

[31] Tibshirani, Robert. "Regression Shrinkage and Selection via the lasso". Journal of the Royal Statistical Society. Series B (methodological). Wiley. 1996, 58(1): 267–88.

[32] Tibshirani, Robert. "The lasso Method for Variable Selection in the Cox Model". Statistics in Medicine. 1997, 16(4): 385–395.

[33] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoCV.html

[34] Hopfield, J. J. "Neural networks and physical systems with emergent collective computational abilities". Proc. Natl. Acad. Sci. U.S.A. 1982, 79(8): 2554–2558.

[35] "Neural Net or Neural Network - Gartner IT Glossary". www.gartner.com.