Quantitative Methods of Data Analysis: Final Project

Equations of State of ε-Iron in the Terrestrial Planetary Interiors

Jingyi Zhuang (uni: jz2907)

Introduction

The thermodynamic properties of hcp-iron (ε-Fe), the likely stable phase of iron at the extreme conditions in the cores of terrestrial planets, are important for understanding planetary interiors. Pioneering works measuring the equations of state (EoS) of ε-Fe include diamond anvil cells (DAC) (Dewaele et al., 2006; Dubrovinsky et al., 2000; Mao et al., 1990) and shock-wave (Brown and Mcqueen, 1986; Yoo et al., 1993) experiments up to 300 GPa, and ab initio calculations using Quantum Monte Carlo (QMC) (Sola et al., 2009), all-electron methods (Sha and Cohen, 2010; Stixrude, 2012), pseudopotential methods (Alfe et al., 2002; Steinle-Neumann et al., 1999; Vočadlo et al., 1997, 2000) and molecular dynamics (MD) (Alfe, 2010; Belonoshko et al., 2011). A recent ramp compression experimental investigation (Smith et al., 2018) of the density-pressure EoS of ε-Fe reached 1.4 TPa under unconstrained temperatures. However, the conditions at planetary interiors include not only high pressures but also high temperatures, i.e. up to 360 GPa and 6,000 K in the Earth's core (Wang et al., 2013). Moreover, recent modeling of terrestrial exoplanets with up to 20 Earth masses (van den Berg et al., 2019) shows that pressures and temperatures at the center of these planets with Earth-like compositions and newly identified phases, can reach over 13 TPa and 9,000 K. These conditions remain challenging as such high temperatures bring extra complexity, e.g., anharmonicity, electronic thermal excitations, and diffusion in computations of solid-state properties. In previous ab initio studies, some or all these effects were partially ignored or completely mingled. In my previous study, I implemented a free energy calculation scheme based on the phonon gas model compatible with temperature-dependent phonon frequencies to investigate ε-Fe at planetary interiors' conditions. The available results include thermodynamic properties such as free energies, bulk modulus, thermal expansivity, and equations of states. This project is going to compare and analyze the ε-Fe compression curves from my previous study and different pieces of literatures.

Method and Objective

The code of this project is using `Python` language, with the help of the package `qha`, `numpy`, `scipy`, `pandas` and `matplotlib`. This project generally includes two parts, with analysis and comparisons of two kinds of equations of states. To briefly describe what methods or algorithms are adopted in this study, the flowchart is presented here to show the organization of data and codes.

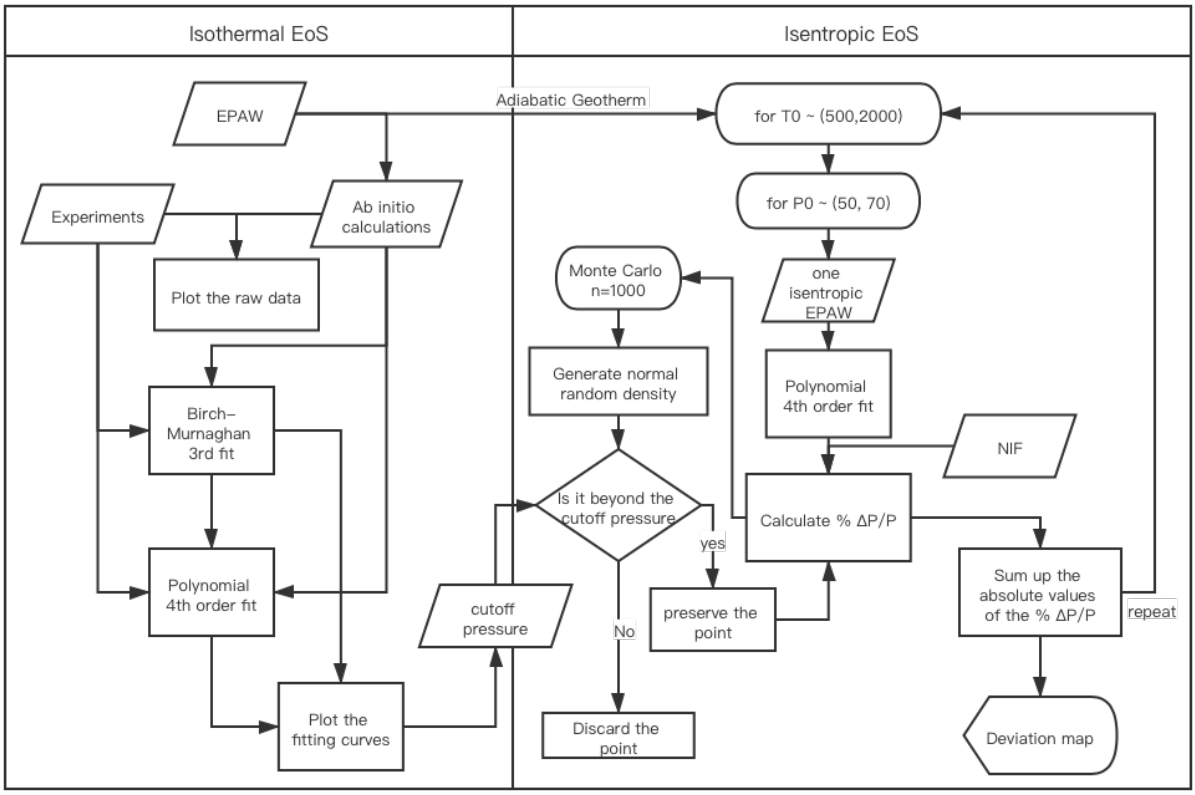

Fig. 1 Flowchart of this project. It starts at the data of EPAW, from top to bottom, and from left to right.

This manuscript will follow this flowchart: firstly, analyze the isothermal equations of state by applying Birch-Murnaghan 3rd order fitting and polynomial 4th order fitting, show the intrinsic difference between experimental data and *ab initio* computational data, obtain the cutoff pressure; then secondly, do the uncertainty exercise by using Monte Carlo method, and plot the deviation between NIF data and multiple EPAW simulations, to pick out the best initial temperature and pressure condition.

The major objective of this project is to practice the data analytical method, like how to fit the data and measure the errors and uncertainty, and evaluate the dataset in a fair and fast way. This practice is aimed to get the insight of how to combine the experimental and ab initio calculation ε-Fe data and, develop the thermodynamics data base, which will aid further modeling terrestrial planetary interior.

Description of data

The data from my previous study used the EPAW method on the first principle (*ab initio*) calculations. It covers a wide temperature range (0 - 8000 K) and pressure range (0 – 1500 GPa), which is able to reach the exoplanetary conditions of up to 5 Earth masses. Aside from the computational tools, we also developed a thermodynamics algorithm, named “Phonon Gas Model” (PGM). It helped generate a series of thermodynamic properties such as free energies, entropies, bulk modulus, thermal expansion and etc. in all expected conditions, i.e., along isotherm or geothermal isentrope. In this project, I am simply

focusing on the compression curves, which are the equations of state, the volume or density vs. pressure. The data will be labeled as 'EPAW' in the following parts.

The first part of this project involves a summary of experimental and computational predicted isotherm equations of state at 300 K, and data from our own study under the same conditions.

We collect the data of compression curves, the density or volume vs. pressure of ε-Fe, from the studies and literatures listed in Table 1. The data are obtained from either *in situ* experiments or *ab initio* calculations. Some of the data sets are reported as scattered points without any fitting or interpolation, while others are reported by fitting parameters, according to the Birch-Murnaghan 3rd order equation of state (BM3):

$$P(V) = \frac{3K_0}{2}\left[\left(\frac{V_0}{V}\right)^{\frac{7}{3}} - \left(\frac{V_0}{V}\right)^{\frac{5}{3}}\right]\left\{1 + \frac{3}{4}(K_0' - 4)\left[\left(\frac{V_0}{V}\right)^{\frac{2}{3}} - 1\right]\right\} \quad \text{Eq. (1)}$$

or the Vinet equation of state (Vinet):

$$P(V) = 3K_0\left(\frac{V}{V_0}\right)^{-\frac{2}{3}}\left[1 - \left(\frac{V}{V_0}\right)^{\frac{1}{3}}\right]\exp\left\{\frac{3}{2}(K_0' - 1)\left[1 - \left(\frac{V}{V_0}\right)^{\frac{1}{3}}\right]\right\} \quad \text{Eq. (2)}$$

The parameters are isothermal bulk modulus K_0, the derivative of bulk modulus with respect to pressure K_0', and the reference volume V_0.

Table 1 Sources of the data of 300 K isotherm compression curve used in this project

Author	Data type	Fitting Eq.	Reference
Mao	*In situ* experiment		(Mao et al., 1990)
Dubrovinsky	*In situ* experiment		(Dubrovinsky et al., 2000)
Dewaele	*In situ* experiment		(Dewaele et al., 2006)
Sakai	*In situ* experiment		(Sakai et al., 2014)
Fei	*In situ* experiment		(Fei et al., 2016)
Yamazaki	*In situ* experiment	BM3	(Yamazaki et al., 2012)
Alfè	*Ab initio* calculation	BM3	(Alfe et al., 2002)
Sola	*Ab initio* calculation		(Sola et al., 2009)
Ono	*Ab initio* calculation	Vinet	(Ono et al., 2010)
Sha	*Ab initio* calculation	Vinet	(Sha and Cohen, 2010)

I will show the *ab initio* calculation results from our own research, with pressures from 0 to 1,500 GPa.

The second part of this project uses the compression curve another recent ramp compression experiment (Smith et al., 2018), which was conducted at the National Ignition Facility (NIF), with the pressure ranging up to 1,400 GPa. It is reported as along the isentrope. This experimental data include uncertainty of both pressure and density.

Isothermal Equations of State

I obtained the ambient pressure-volume equations of state through *ab initio* calculations using EPAW method and compared with the data listed in Table 1.

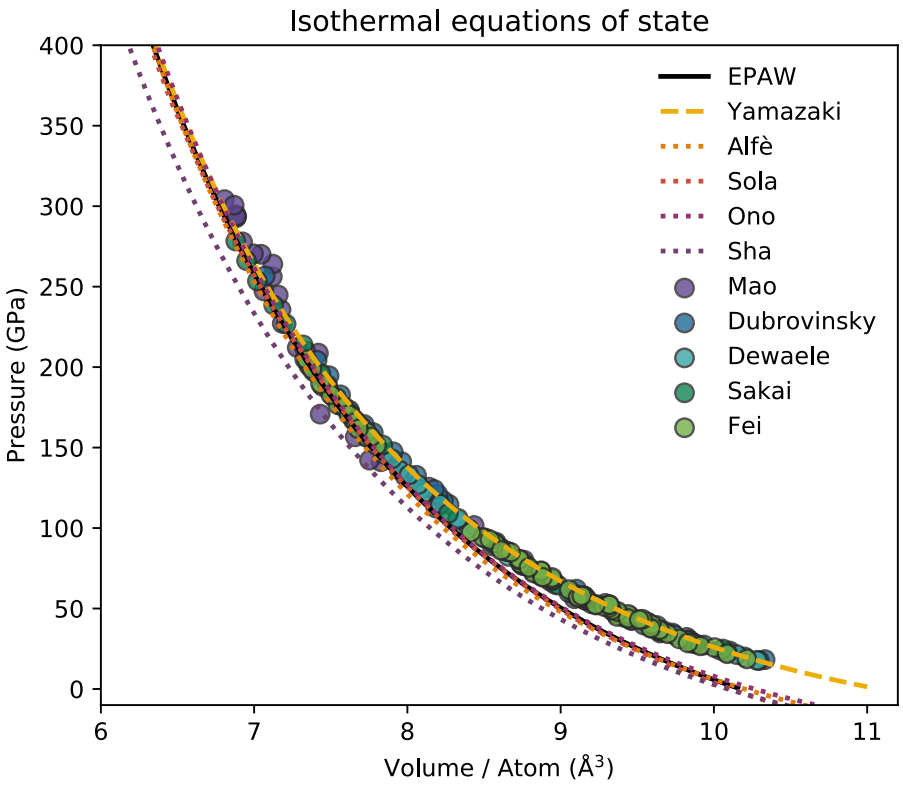

Fig. 2 Isothermal equations of state of the *in situ* experiments and *ab initial* experiments. The data providing fitting parameters are plotted as a continuous line with the volume varies from 6 to 11 Å³/atom.

Before any further statistical analysis, it is obvious that the EPAW agrees well with other *ab initio* predictions but not with experimental measurements in the relevant low-pressure range. Also, we can see that there are differences between experimental data: though most of them using diamond anvil cell methods, they are not all consistent with each other. The discrepancy increases in higher pressures, while the theoretical predictions behave better at higher pressures.

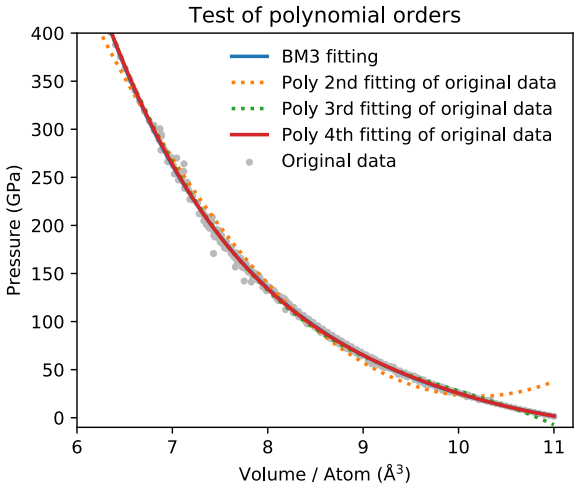

Fig. 3 Test of different polynomial orders, n=2, 3, 4, of the experimental data sets. It shows that n=2 and n=3 do not work well because they have obvious deviation.

However, the understanding of "low pressure validity" is important to select the proper valid region to do the following uncertainty excises.

As the shape of the compression curve owns important physical properties of bulk modulus and etc., I tried to fit the curve firstly by using Eq. (1). The obtained BM3 fitting parameter is using the 'curve_fit' function from the python package `scipy`. Then, I applied polynomial 4th order fitting on the original data points, as well as on the BM3 fitting curve. The fitting curve comparison of Fig. 4 shows that the three fitting methods do not deviate much from each other. The polynomial 4th order fitting of original data shows good consistency with the BM3 fitting curve.

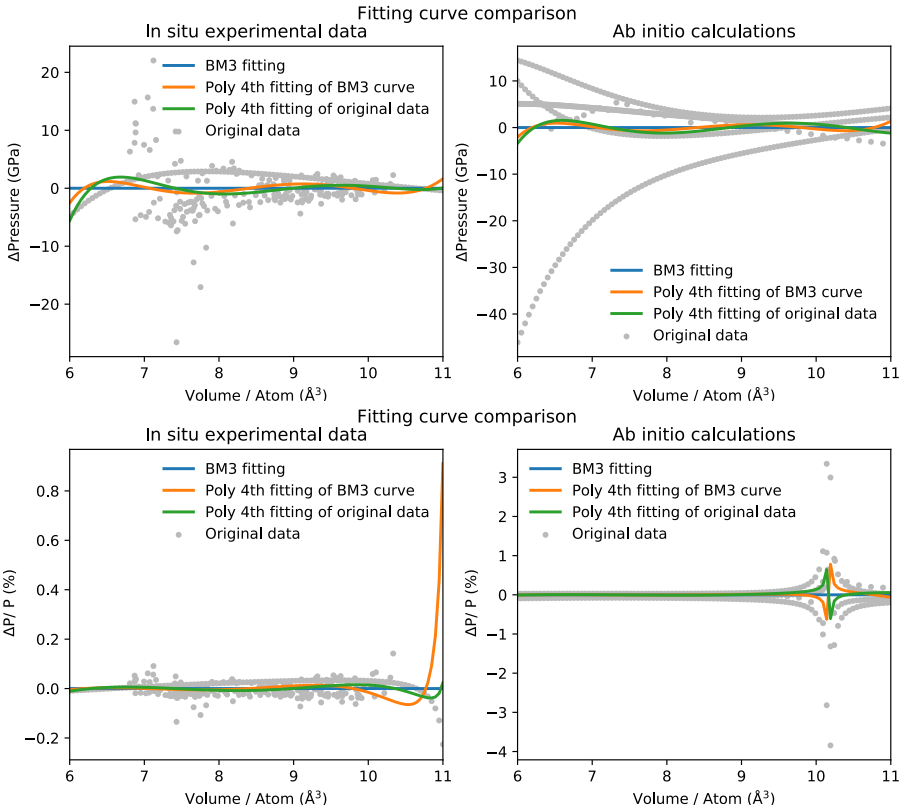

Fig. 4 Fitting of EoS from experiments (left) and ab initio calculations (right). The upper set shows ΔP, which is the difference between the 'standard' BM3 fitting results and the calculated pressures of each curve. The lower set shows $\Delta P \,/\, P_{\mathrm{BM3}}$.

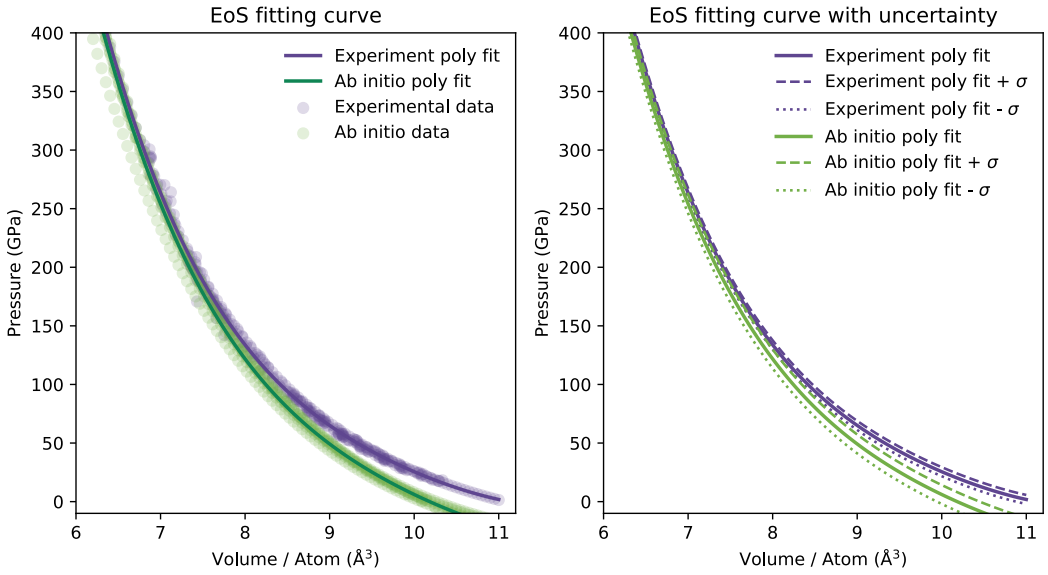

Fig. 5 Equations of state fitting curve and the comparison with the raw data (left) and with data errors σ_e (right). The convergence at higher pressures is good.

The fitting curves of the equations of state are plotted with their data errors. Essentially all ab initio predictions show very good agreement with the experimental data above 100 GPa from Fig. 5, which is the pressure range of concern, as I focus on much higher pressure. This is a very important point when considering the state of iron in the Earth' solid inner core (beyond 330 GPa) and in the core of terrestrial exoplanets (beyond one to several thousands of GPa). This discrepancy is reasonable due to several reasons: 1) the non-magnetism of *ab initio* calculation, while at low pressure it could have magnetism; 2) the unspecified phase, since ε phase of Fe does not exist at low pressures.

I carefully choose $P_{cut} = 100$ GPa for the next step, to discard data points below this cutoff pressure, in order to fairly compare the isentropic results from our study with the NIF experiments.

Isentropic Equations of State

In my study, the isentropic equations of state are calculated along the adiabatic geotherm,

$$\left(\frac{\partial T}{\partial P}\right)_S = \frac{\alpha V T}{C_p} \qquad \text{Eq. (3)}$$

And then do the integration from appropriate initial conditions T_0 and P_0,

$$T = T_0 + \int_{P_0}^{P} \left(\frac{\partial T}{\partial P}\right)_S dP \qquad \text{Eq. (4)}$$

Thus, the choice of different initial conditions (T_0, P_0), will affect my simulated results.

In the NIF experiment (Smith et al., 2018), the initial pressure was reported as 60 GPa, while the initial temperature remained uncertain since it cannot be measured directly. The possible temperature range was believed to be around or beyond 1000 K. Thus, we are taking the first adiabatic guess with initial conditions of $T_0 = 1000$ K and $P_0 = 60$ GPa. Due to the pre-existing precision issue, the density vs pressure curve from EPAEW is produced with step-like shape (physically and mathematically it should not have been like this), which is shown in Fig. 6.

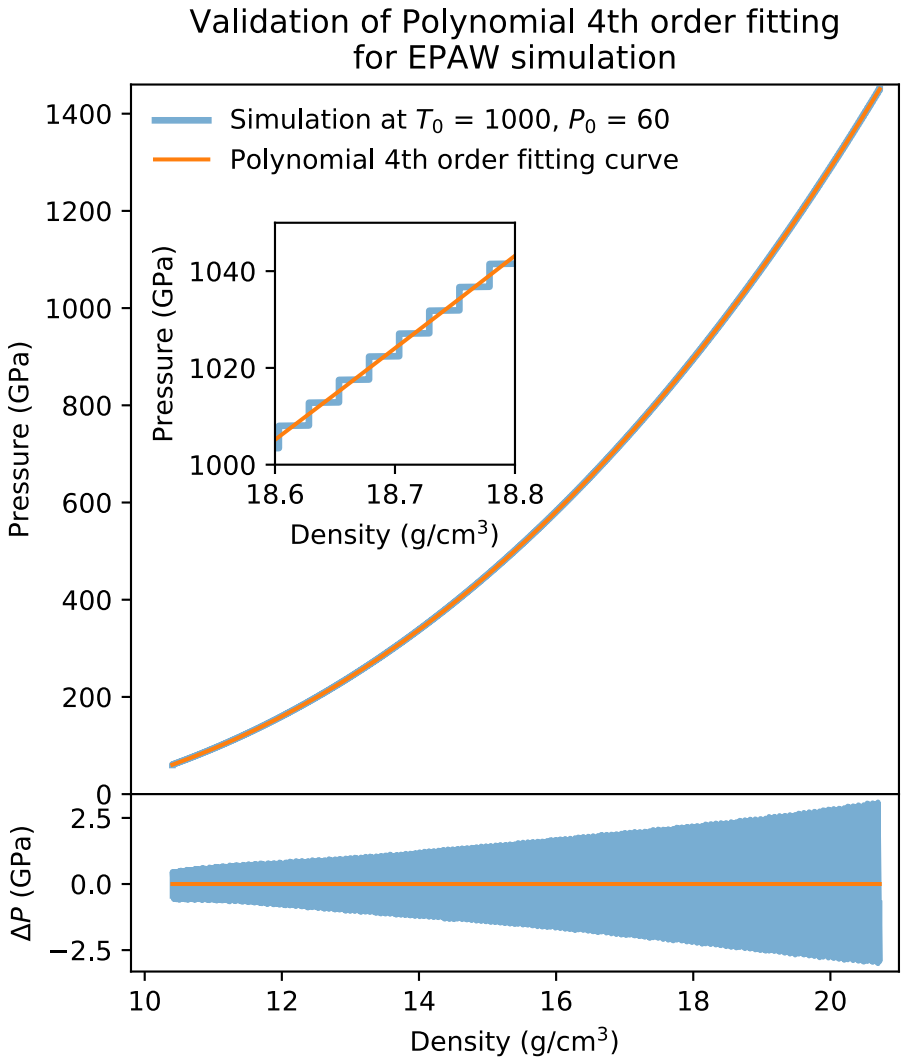

Fig. 6 Validation of polynomial 4th order fitting curve. The inserted plot shows the step-like shape due to the pre-existing precision issue. The bottom plot shows the difference between the original data and the fitted curve.

The polynomial 4th order fitting curve well reproduces the EPAW and smoothens the step-like fluctuations. Thus, the following simulation curves under different temperature and pressure conditions are fitted by polynomial 4th order to fasten the calculation.

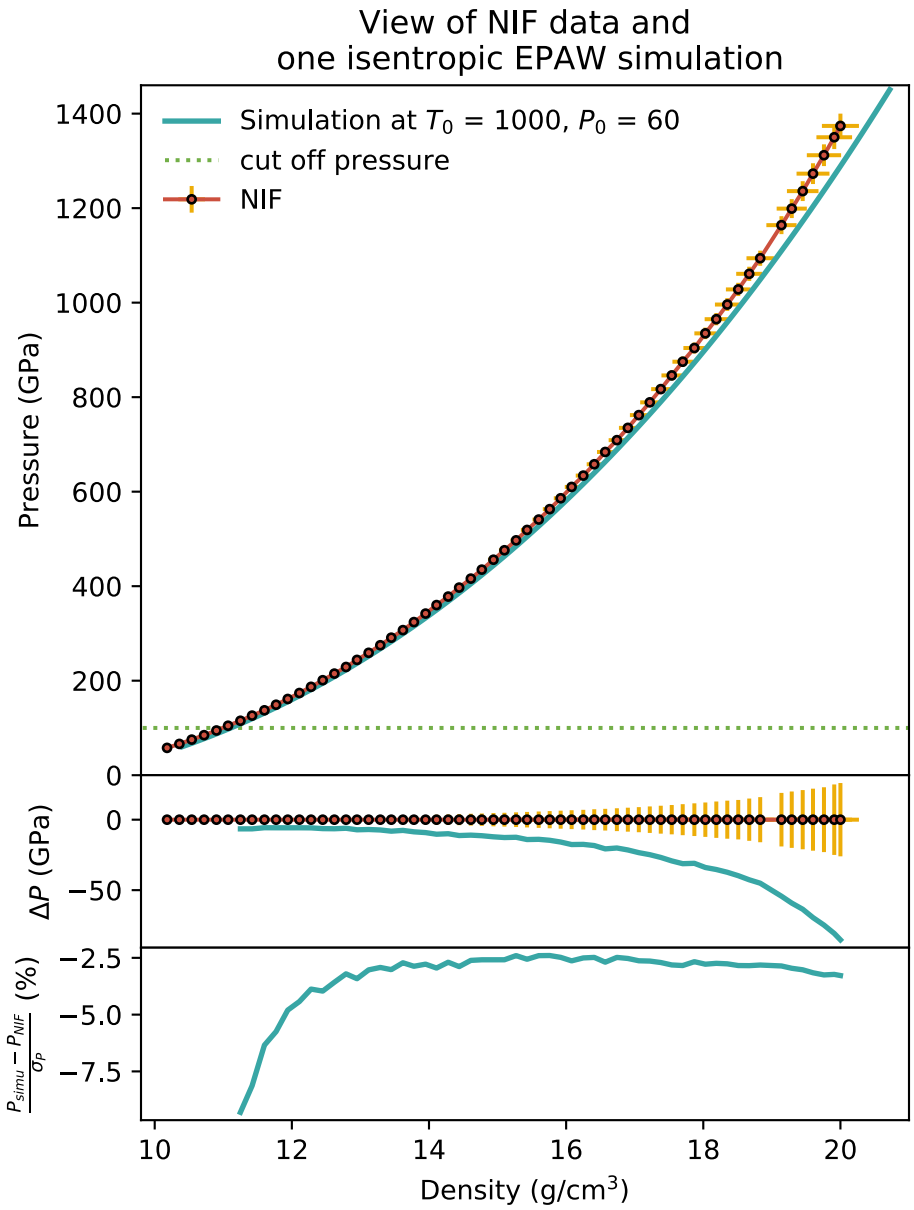

Fig. 7 View of NIF data and one isentropic simulation under the experiment's favored initial conditions. The bottom two figure shows the pressure difference compared to NIF's pressure uncertainty by using the Monte Carlo density propagation uncertainty analysis.

To analyze the behavior of our simulations, we are calculating the pressure difference $\Delta P = P_{simu} - P_{NIF}$ and the corresponding difference ratio $\frac{\Delta P}{\sigma_{P_{NIF}}}$, discarding the data points under the cutoff pressure, which is defined in the isotherm part. To properly take the density uncertainty into consideration, I am using the Monte Carlo method. Within each Monte Carlo iteration, I obtain the 'psedo' data point by

taking the random number using normal distribution, with the mean value to be NIF's density, $\mu = d_{NIF}$, and the standard deviation to be half of the NIF's density uncertainty, $\sigma = \sigma_{NIF}/2$. Because the paper did not specify the type of uncertainties it reported, I assume this uncertainty is 2σ of normal distribution. Then I can obtain the pressure difference ratios for each loop, and calculate the average of the ratios for the whole Monte Carlo calculations.

I extracted the EPAW isentropic simulations with a series of initial conditions, of T_0 in the range from 500 to 2000 K with the interval of 100K, and P_0 in the range from 50 to 70 GPa with the interval of 1 GPa. For each simulation, it starts at one initial condition, calculates the adiabatic temperature and pressure relation, and reproduces the compression curve with respect to its T-P profile. The total deviation of such single simulation is calculated by $D = \sum \left| \frac{\Delta P}{\sigma_{P_{NIF}}} \right|$. Bring all 336 simulations together, the deviation quantities can visualize and reveal the computational preferred initial conditions.

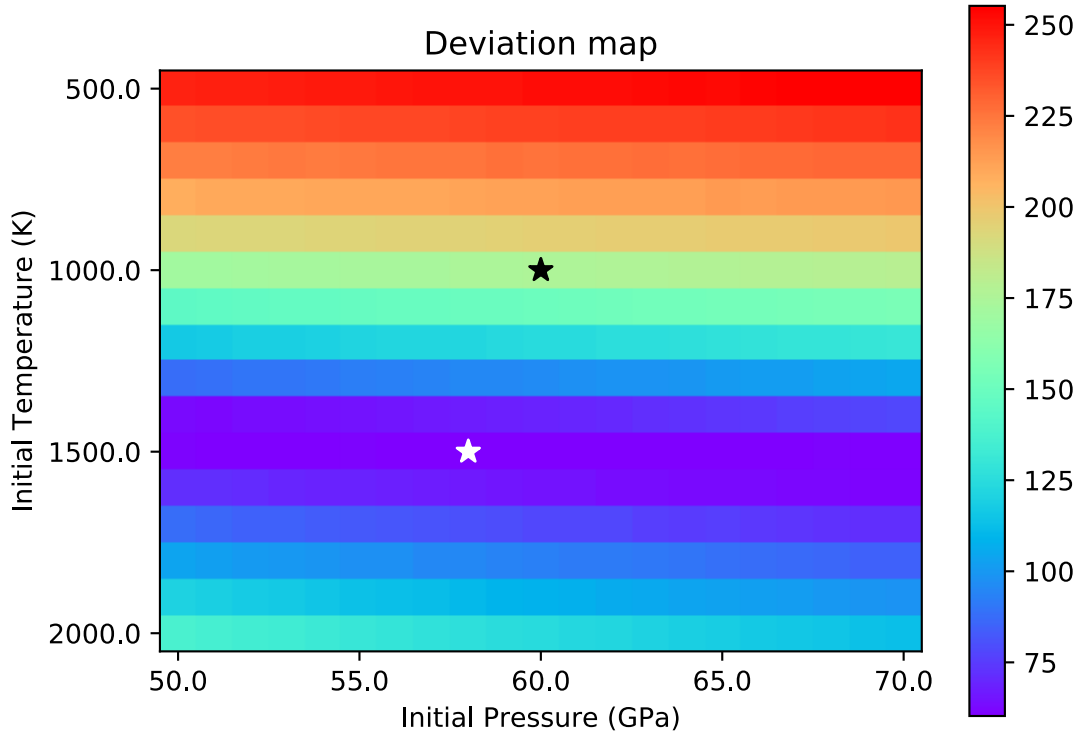

Fig. 8 Deviation plot. The black star stands for the experimental favored condition. The white star shows the calculated minimum deviation.

It is interesting to observe that the initial temperature of the adiabatic temperature-pressure relation will affect much more than the initial pressure. According to Fig. 8, the minimum deviation lies at $T_0 = 1500$ K, $P_0 = 58$ GPa, which is a possible condition regarding the NIF paper. Since we get a reliable initial condition, we can deduct the related temperature and pressure profile, and calculate all the thermodynamic properties along the profile, e.g. thermal expansivity, bulk modulus, thermal conductivity and etc., as a part of the future work, to model the thermal conduction of inner cores.

Conclusions

Through this study of equations of state, several tests show that the polynomial 4th order fitting is good enough to reproduce the compression curve and almost equivalent to Birch-Murnaghan 3rd equation when we are not doing any extrapolation. The intrinsic discrepancies between in situ experiments and ab initio calculations are displayed by the fitting curves and deviations, and they show that the ab initio results are more reliable in the high-pressure region rather than the low-pressures. The Monte Carlo tests simplify the uncertainty lying on both x- and y- axis into the more familiar y-axis uncertainty, and help us analyze the behavior of my EPAW isentropic simulations. As the NIF experiment was not able to tell exactly the initial experimental condition, using the *ab initio* simulation results can be a reasonable and reliable method to determine the most likely initial condition. The EPAW and PGM method from my previous research will aid planetary modeling with hot iron cores.

Acknowledge

The Evolutionary PAW (EPAW) method and phonon gas model (PGM) method have been presented on:

1. Thermal equation of state of ε-Fe at exoplanetary interior conditions (oral), American Physical Society (APS) March Meeting, Boston, MA, USA, Mar. 2019.

2. Thermodynamic properties of ε-Fe at inner core conditions using T-dependent phonon dispersions (poster), American Geophysical Union (AGU) Fall Meeting, Washington DC, Dec. 2018.

The data of EPAW involved in this manuscript are part of these presentations. The data analysis, e.g. fitting curve comparisons and Monte Carlo uncertainty calculations, are completely original and nonrelated to the previous presentations.

Reference

Alfe, D. (2010). Iron at Earth's Core Conditions from First Principles Calculations. Reviews in Mineralogy and Geochemistry *71*, 337–354.

Alfe, D., Gillan, M.J., Vocadlo, L., Brodholt, J., and Price, G.D. (2002). The ab initio simulation of the Earth's core. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences *360*, 1227–1244.

Belonoshko, A.B., Arapan, S., and Rosengren, A. (2011). An ab initio molecular dynamics study of iron phases at high pressure and temperature. Journal of Physics Condensed Matter *23*.

van den Berg, A.P., Yuen, D.A., Umemoto, K., Jacobs, M.H.G., and Wentzcovitch, R.M. (2019). Mass-dependent dynamics of terrestrial exoplanets using ab initio mineral properties. Icarus.

Brown, J.M., and Mcqueen, G. (1986). Phase Transitions , Griineisen Parameter , and Elasticity. Journal of Geophysical Research *91*, 7485–7494.

Dewaele, A., Loubeyre, P., Occelli, F., Mezouar, M., Dorogokupets, P.I., and Torrent, M. (2006). Quasihydrostatic Equation of State of Iron above 2 Mbar. Physical Review Letters *97*.

Dubrovinsky, L.S., Saxena, S.K., Tutti, F., Rekhi, S., and LeBehan, T. (2000). In Situ X-Ray Study of Thermal Expansion and Phase Transition of Iron at Multimegabar Pressure. Physical Review Letters *84*, 1720–1723.

Fei, Y., Murphy, C., Shibazaki, Y., Shahar, A., and Huang, H. (2016). Thermal equation of state of hcp-iron: Constraint on the density deficit of Earth's solid inner core. Geophysical Research Letters *43*, 6837–6843.

Mao, H.K., Wu, Y., Chen, L.C., and Shu, J.F. (1990). Static compression of iron to 300 GPa and Fe(0.8)Ni(0.2) alloy to 260 GPa - Implications for composition of the core. Journal of Geophysical Research *95*, 21737.

Ono, S., Kikegawa, T., Hirao, N., and Mibe, K. (2010). High-pressure magnetic transition in hcp-Fe. American Mineralogist *95*, 880–883.

Sakai, T., Takahashi, S., Nishitani, N., Mashino, I., Ohtani, E., and Hirao, N. (2014). Equation of state of pure iron and Fe0.9Ni0.1alloy up to 3Mbar. Physics of the Earth and Planetary Interiors *228*, 114–126.

Sha, X., and Cohen, R.E. (2010). First-principles thermal equation of state and thermoelasticity of hcp Fe at high pressures. Physical Review B - Condensed Matter and Materials Physics *81*, 1–33.

Smith, R.F., Fratanduono, D.E., Braun, D.G., Duffy, T.S., Wicks, J.K., Celliers, P.M., Ali, S.J., Fernandez-Pañella, A., Kraus, R.G., Swift, D.C., et al. (2018). Equation of state of iron under core conditions of large rocky exoplanets. Nature Astronomy *2*, 452–458.

Sola, E., Brodholt, J.P., and Alfè, D. (2009). Equation of state of hexagonal closed packed iron under Earth's core conditions from quantum Monte Carlo calculations. Physical Review B - Condensed Matter and Materials Physics *79*, 1–6.

Steinle-Neumann, G., Stixrude, L., and Cohen, R.E. (1999). First-principles elastic constants for the hcp transition metals Fe, Co, and Re at high pressure. Physical Review B - Condensed Matter and Materials Physics *60*, 791–799.

Stixrude, L. (2012). Structure of Iron to 1 Gbar and 40 000 K. Physical Review Letters *108*.

Vočadlo, L., de Wijs, G.A., Kresse, G., Gillan, M., Price, G.D., Vocadlo, L., and de Wijs, G.A. (1997). First principles calculations on crystalline and liquid iron at Earth's core conditions. Faraday Discussions *106*, 205–218.

Vočadlo, L., Brodholt, J., Alfè, D., Gillan, M.J., and Price, G.D. (2000). Ab initio free energy calculations on the polymorphs of iron at core conditions. Physics of the Earth and Planetary Interiors *117*, 123–137.

Wang, J., Smith, R.F., Eggert, J.H., Braun, D.G., Boehly, T.R., Reed Patterson, J., Celliers, P.M., Jeanloz, R., Collins, G.W., and Duffy, T.S. (2013). Ramp compression of iron to 273 GPa. Journal of Applied Physics *114*.

Yamazaki, D., Ito, E., Yoshino, T., Yoneda, A., Guo, X., Zhang, B., Sun, W., Shimojuku, A., Tsujino, N., Kunimoto, T., et al. (2012). P-V-T equation of state for ϵ-iron up to 80 GPa and 1900 K using the Kawai-type high pressure apparatus equipped with sintered diamond anvils. Geophysical Research Letters *39*, 1–6.

Yoo, C.S., Holmes, N.C., Ross, M., Webb, D.J., and Pike, C. (1993). Shock temperatures and melting of iron at Earth core conditions. Physical Review Letters *70*, 3931–3934.

Appendix: Code

Isothermeos.py

```
'''
This python code is used to draw the graph of isothermal Equations of states.
author: jz2907@columbia.edu
'''

from scipy.optimize import curve_fit
import numpy as np
from numpy.linalg import inv
import pandas as pd
import os
import matplotlib.pyplot as plt
import palettable

from matplotlib.axes._axes import _log as matplotlib_axes_logger
matplotlib_axes_logger.setLevel('ERROR')

# Color initialize
cm = palettable.cartocolors.qualitative.Prism_10
color_list = cm.mpl_colors
color_list.insert(0,"black")
color_exp = color_list[1]
color_ab = color_list[5]

def readcsv(fileName, folderName="data"):
    '''
    read the compression curve data, default folder '/data'
    return volume, pressure, in unit of A3 and GPa respectively.
    '''
    df = pd.read_csv(os.path.join(folderName, fileName))
    v = df['v']
    p = df['p']
    return v, p

def vinet_p(v, k0, kp0, v0):
    '''
    P(V) of Vinet equations of state.
    return pressures
    '''
    x = (v / v0) ** (1 / 3)
    xi = 3 / 2 * (kp0 - 1)
    return 3 * k0 / x ** 2 * (1 - x) * np.exp(xi * (1 - x))

def vp_vinet(k0, kp0, v0):
    '''
```

```
    give the fitting parameters k0, kp0, and v0,
    the default setting of volume is 6 ~ 11 A3
    return the volumes and Vinet pressures.
    '''
    v = np.linspace(6, 11, 100)
    return v, vinet_p(v, k0, kp0, v0)

def bm3_p(v, k0, kp0, v0):
    '''
    P(V) of Birch-Murnaghan 3rd order equations of state.
    return pressures
    '''
    eta = (v0 / v) ** (1 / 3)
    return 3 / 2 * k0 * (eta ** 7 - eta ** 5) * (1 + 3 / 4 * (kp0 - 4) * (eta ** 2 -
1))

def vp_bm3(k0, kp0, v0):
    '''
    give the fitting parameters k0, kp0, and v0,
    the default setting of volume is 6 ~ 11 A3
    return the volumes and Birch-Murnaghan 3rd order pressures.
    '''
    v = np.linspace(6,11,100)
    return v, bm3_p(v,k0,kp0,v0)

def eos_fit(v: np.ndarray, p: np.ndarray):
    '''
    return popt, pcov of the fitting parameters;
    popt is (k0, kp0, v0)
    '''
    return curve_fit(bm3_p, v, p, p0=[200, 4, 10])

def poly_calc(x, param):
    '''
    return the poly 2nd order result: a + b x + c x^2
    '''
    ans = 0 * x
    for i,a in enumerate([*param]):
        ans += a * x ** i
    return ans

def poly_fit(v, p, n = 4):
    '''
    polynomial fitting a + b x + c x^2 + ...
    default n = 4
    return a,b,c,...
    '''
```

```
    G = np.ones((len(v),n+1))
    for i in range(n):
        G[:,i+1] = v **(i+1)
    Gt = np.transpose(G)
    invGtG = inv(np.matmul(Gt,G))
    tov = np.matmul(invGtG, Gt)
    mest = np.matmul(tov, p)

    # calculate the sig_m
    e = p - np.matmul(G, mest)
    et = np.transpose(e)
    E = np.matmul(et, e)
    sigma2d = E / (len(p) - (n+1))
    Cm = sigma2d * invGtG
    sigma2m = np.diag(Cm)
    # print(sigma2m**0.5,mest)

    return mest, sigma2d**0.5

def plot_raw_data(data):
    '''
    draw all the data passing into this function
    The form of data should be:
    data = [ dict1, dict2, ... ]
    and the dicts should be:
    dict = {
            "v" = [...],
            "p" = [...],
            "draw" = "plot" or "scatter",
            "color" = "...",
            "label" = "...",
            "ls" = "the_line_style"
    }
    '''

    plt.figure(figsize=(6,5))

    for item in data:
        if item["draw"] == "plot":

plt.plot(item["v"],item["p"],c=item["color"],label=item["label"],lw=2,ls=item["ls"])
        elif item["draw"] == "scatter":
            plt.scatter(item["v"], item["p"], c=item["color"], label=item["label"],
lw=1,s=60, marker=item["ls"],edgecolors="#2B2B2B",alpha=0.8)

    plt.xlabel('Volume / Atom (Å$^3$)')
```

```
    plt.ylabel('Pressure (GPa)')

    plt.xlim(6, 11.2)
    plt.ylim(-10, 400)

    plt.title("Isothermal equations of state")

    plt.legend(frameon=False)
    plt.savefig("graph/300K.pdf", dpi=300)

def plot_the_data(data):
    data_p = [(i["p"]) for i in data]
    data_p = np.concatenate(data_p)
    data_v = [(i["v"]) for i in data]
    data_v = np.concatenate(data_v)

    if data[0]["type"] == "exp":
        label = "Experimental data"
        color = color_exp
    else:
        label = "Ab initio data"
        color = color_ab
    plt.scatter(data_v, data_p,
                marker="o", label=label, c=color, alpha = 0.2)

def plot_poly_fitting(data):
    '''
    '''
    data_p = [(i["p"]) for i in data]
    data_p = np.concatenate(data_p)
    data_v = [(i["v"]) for i in data]
    data_v = np.concatenate(data_v)

    # calculate the bm3 fit parameters of data
    data_bm3_fit, _ = eos_fit(data_v, data_p)
    data_bm3_fit_v, data_bm3_fit_p = vp_bm3(*data_bm3_fit)

    # calculate the 4nd order poly fit of the origional data set
    data_poly_fit,data_poly_fit_sig = poly_fit(data_v, data_p)
    data_poly_fit_p = poly_calc(
        data_bm3_fit_v, data_poly_fit)

    # calculate the 4nd order polynomial fit using least sqare solution of the bm3
fitted curve
    data_bm3_fit_poly_fit, data_bm3_fit_poly_fit_sig = poly_fit(
        data_bm3_fit_v, data_bm3_fit_p)
    data_bm3_fit_poly_fit_p = poly_calc(
```

```
        data_bm3_fit_v, data_bm3_fit_poly_fit)

    # plot of fitting curves

    plt.plot(data_bm3_fit_v, data_bm3_fit_p -
             data_bm3_fit_p, label="BM3 fitting",lw=2)
    plt.plot(data_bm3_fit_v, data_bm3_fit_poly_fit_p -
             data_bm3_fit_p, label="Poly 4th fitting of BM3 curve",lw=2)
    plt.plot(data_bm3_fit_v, data_poly_fit_p-data_bm3_fit_p,
             label="Poly 4th fitting of original data",lw=2)
    plt.scatter(data_v, data_p-bm3_p(data_v, *data_bm3_fit),
                marker=".", label="Original data",c="#BABABA")

    # plt.plot(data_bm3_fit_v, (data_bm3_fit_p -
    #                           data_bm3_fit_p)/data_bm3_fit_p, label="BM3 fitting",
lw=2)
    # plt.plot(data_bm3_fit_v, (data_bm3_fit_poly_fit_p -
    #                           data_bm3_fit_p)/data_bm3_fit_p, label="Poly 4th
fitting of BM3 curve", lw=2)
    # plt.plot(data_bm3_fit_v, (data_poly_fit_p-data_bm3_fit_p)/data_bm3_fit_p,
    #          label="Poly 4th fitting of original data", lw=2)
    # plt.scatter(data_v, (data_p-bm3_p(data_v, *data_bm3_fit))/bm3_p(data_v,
*data_bm3_fit),
    #             marker=".", label="Original data", c="#BABABA")

    plt.xlabel('Volume / Atom (Å$^3$)')
    plt.ylabel('$\\Delta$Pressure (GPa)')
    plt.xlim(6, 11)
    if data[0]["type"] =="exp":
        plt.title("In situ experimental data")
    else:
        plt.title("Ab initio calculations")
    plt.legend(frameon=False)

    return data_poly_fit, data_poly_fit_sig

def plot_poly_fitting_veri(data):
    '''
    '''
    data_p = [(i["p"]) for i in data]
    data_p = np.concatenate(data_p)
    data_v = [(i["v"]) for i in data]
    data_v = np.concatenate(data_v)

    # calculate the bm3 fit parameters of data
    data_bm3_fit, _ = eos_fit(data_v, data_p)
    data_bm3_fit_v, data_bm3_fit_p = vp_bm3(*data_bm3_fit)
```

```
    # calculate the 2nd order poly fit of the origional data set
    data_poly_2_fit, _ = poly_fit(data_v, data_p,n=2)
    data_poly_2_fit_p = poly_calc(
        data_bm3_fit_v, data_poly_2_fit)

    # calculate the 3nd order poly fit of the origional data set
    data_poly_3_fit, _ = poly_fit(data_v, data_p,n=3)
    data_poly_3_fit_p = poly_calc(
        data_bm3_fit_v, data_poly_3_fit)

    # calculate the 4nd order poly fit of the origional data set
    data_poly_4_fit, _ = poly_fit(data_v, data_p)
    data_poly_4_fit_p = poly_calc(
        data_bm3_fit_v, data_poly_4_fit)


    # plot of fitting curves

    plt.plot(data_bm3_fit_v, data_bm3_fit_p, label="BM3 fitting", lw=2)
    plt.plot(data_bm3_fit_v, data_poly_2_fit_p,
             label="Poly 2nd fitting of original data", lw=2, ls=":")
    plt.plot(data_bm3_fit_v, data_poly_3_fit_p,
             label="Poly 3rd fitting of original data", lw=2,ls=":")
    plt.plot(data_bm3_fit_v, data_poly_4_fit_p,
             label="Poly 4th fitting of original data", lw=2, ls="-")

    plt.scatter(data_v, data_p,
                marker=".", label="Original data", c="#BABABA")
    plt.xlabel('Volume / Atom (Å$^3$)')
    plt.ylabel('Pressure (GPa)')
    plt.xlim(6, 11)
    plt.legend(frameon=False)

if __name__ == "__main__":

    #-----------------------------------------------------#
    #                 initialize data                     #
    #-----------------------------------------------------#
    data = []

    # EPAW data
    epaw_v ,epaw_p = readcsv("epawmethod.csv")
    epaw = {"v":epaw_v[55:250],
            "p": epaw_p[55:250],
            "type": "ab",
            "draw":"plot",
```

```
            "ls":"solid",
            "label":"EPAW"}
    data.append(epaw)

    # Mao data, experiment
    mao_v, mao_p = readcsv("mao.csv")
    mao = {"v": mao_v,
            "p": mao_p,
            "type": "exp",
            "draw": "scatter",
            "ls": "o",
            "label": "Mao"}
    data.append(mao)

    # Dubrovinsky data, experiment
    dub_v, dub_p = readcsv("dub300.csv")
    dub = {"v": dub_v,
           "p": dub_p,
           "type": "exp",
           "draw": "scatter",
           "ls": "o",
           "label": "Dubrovinsky"}
    data.append(dub)

    # Dewaele data, experiment
    dewaele_v, dewaele_p = readcsv("dewaele.csv")
    dewaele = {"v": dewaele_v,
               "p": dewaele_p,
                "type": "exp",
                "draw": "scatter",
                "ls": "o",
                "label": "Dewaele"}
    data.append(dewaele)

    # Sakai data, experiment
    sakai_v, sakai_p = readcsv("sakai.csv")
    sakai = {"v": sakai_v,
             "p": sakai_p,
              "type": "exp",
              "draw": "scatter",
              "ls": "o",
              "label": "Sakai"}
    data.append(sakai)

    # Fei data, experiment
    fei_v, fei_p = readcsv("fei.csv")
    fei = {"v": fei_v,
```

```
          "p": fei_p,
           "type": "exp",
           "draw": "scatter",
           "ls": "o",
           "label": "Fei"}
    data.append(fei)

    # Yamazaki data, experiment, BM3 fitting
    Yamazaki_3BM_V0 = 22.15/2
    Yamazaki_3BM_K0 = 202
    Yamazaki_3BM_Kp = 4.5
    yamazaki_v, yamazaki_p = vp_bm3(Yamazaki_3BM_K0, Yamazaki_3BM_Kp, Yamazaki_3BM_V0)
    yamazaki = {"v": yamazaki_v,
                "p": yamazaki_p,
                "type": "exp",
                "draw": "plot",
                "ls": "--",
                "label": "Yamazaki"}
    data.append(yamazaki)

    # Alfe data, ab initio calculation, BM3 fitting
    Alfe_3BM_V0 = 10.20
    Alfe_3BM_K0 = 291
    Alfe_3BM_Kp = 4.4
    alfe_v, alfe_p = vp_bm3( Alfe_3BM_K0, Alfe_3BM_Kp, Alfe_3BM_V0)
    alfe = {"v": alfe_v,
            "p": alfe_p,
            "type": "ab",
            "draw": "plot",
            "ls": ":",
            "label": "Alfè"}
    data.append(alfe)

    # Sola data, ab initio calculation,
    sola_v, sola_p = readcsv("qmc.csv")
    sola = {"v": sola_v,
           "p": sola_p,
            "type": "ab",
           "draw": "plot",
           "ls": ":",
           "label": "Sola"}
    data.append(sola)

    # Ona data, ab initio calculation, Vinet fitting
    Ono_Vinet_V0 = 10.265
    Ono_Vinet_K0 = 285
    Ono_Vinet_Kp = 4.8
```

```
ono_v, ono_p = vp_vinet(Ono_Vinet_K0, Ono_Vinet_Kp, Ono_Vinet_V0)
ono = {"v": ono_v,
       "p": ono_p,
       "type": "ab",
        "draw": "plot",
        "ls": ":",
        "label": "Ono"}
data.append(ono)

# Sha data, ab initio calculation,  BM3 fitting
Cohen_Vinet_V0 = (20.18/2)
Cohen_Vinet_K0 = 296
Cohen_Vinet_Kp = 4.4
sha_v, sha_p = vp_vinet(Cohen_Vinet_K0, Cohen_Vinet_Kp, Cohen_Vinet_V0)
sha = {"v": sha_v,
       "p": sha_p,
       "type": "ab",
       "draw": "plot",
       "ls": ":",
       "label": "Sha"}
data.append(sha)

# set up the color value
for i, item in enumerate(data):
    item["color"] = color_list[i]

# call the plot function to obtain the graph of raw data
plot_raw_data(data)

#-----------------------------------------------------#
#                   fit the data                      #
#-----------------------------------------------------#

# seperate the experiments and ab initio calculations into two:
experiments = [ i for i in data if i["type"]=="exp"]
abinitio = [(i) for i in data if (i["type"] == "ab")]

vx = np.linspace(6, 11, 100)

# plot the test of polynomial fit & bm3 fit
plt.close()
plt.figure(figsize=(10,3.8))
plt.subplot(1,2,1)
experiments_poly_fit, experiments_poly_fit_sige = plot_poly_fitting(
    experiments)
plt.subplot(1, 2, 2)
abinitio_poly_fit, abinitio_poly_fit_sige = plot_poly_fitting(
```

```
        abinitio)
    plt.suptitle("Fitting curve comparison")
    plt.savefig("graph/300K_diff.pdf", dpi=300)

    # plot the comparison between experiments and ab initio
    plt.close()
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.plot(vx, poly_calc(vx, experiments_poly_fit),
             label="Experiment poly fit", c=color_exp, lw=2)
    plt.plot(vx, poly_calc(vx, abinitio_poly_fit),
             label="Ab initio poly fit", c=color_list[4], lw=2)
    plot_the_data(experiments)
    plot_the_data(abinitio)

    plt.xlabel('Volume / Atom (Å$^3$)')
    plt.ylabel('Pressure (GPa)')
    plt.xlim(6, 11.2)
    plt.ylim(-10, 400)
    plt.title("EoS fitting curve")
    plt.legend(frameon=False)

    plt.subplot(1, 2, 2)
    plt.plot(vx, poly_calc(vx, experiments_poly_fit),
             c=color_exp, label="Experiment poly fit", lw=2)
    plt.plot(vx, poly_calc(vx, experiments_poly_fit) +
             experiments_poly_fit_sige, ls="--", label="Experiment poly fit +
$\\sigma$", c=color_exp)
    plt.plot(vx, poly_calc(vx, experiments_poly_fit) -
             experiments_poly_fit_sige, ls=":", label="Experiment poly fit -
$\\sigma$", c=color_exp)

    plt.plot(vx, poly_calc(vx, abinitio_poly_fit),  c=color_ab, label="Ab initio poly
fit",lw = 2)
    plt.plot(vx, poly_calc(vx, abinitio_poly_fit) +
             abinitio_poly_fit_sige, ls="--", label="Ab initio poly fit + $\\sigma$",
c=color_ab)
    plt.plot(vx, poly_calc(vx, abinitio_poly_fit) -
             abinitio_poly_fit_sige, ls=":", label="Ab initio poly fit - $\\sigma$",
c=color_ab)

    plt.xlabel('Volume / Atom (Å$^3$)')
    plt.ylabel('Pressure (GPa)')
    plt.xlim(6, 11.2)
    plt.ylim(-10, 400)
    plt.title("EoS fitting curve with uncertainty")
    plt.legend(frameon=False)
```

```
plt.savefig("graph/300K_fitting.pdf" , dpi=300)

#-----------------------------------------------------#
#                 why is poly 4th                     #
#-----------------------------------------------------#

plt.close()
plt.figure(figsize=(5, 4))
plot_poly_fitting_veri(experiments)
plt.xlabel('Volume / Atom (Å$^3$)')
plt.ylabel('Pressure (GPa)')
plt.xlim(6, 11.2)
plt.ylim(-10, 400)
plt.title("Test of polynomial orders")
plt.legend(frameon=False)
plt.savefig("graph/300K_poly4why.pdf", dpi=300)
```

nifeos.py

```
'''
This python code is used to draw the graph of isentropic Equations of states, compare
EPAW with NIF.
author: jz2907@columbia.edu
'''

from qha.unit_conversion import *
from scipy.optimize import curve_fit
import numpy as np
from numpy.linalg import inv
import pandas as pd
import os
import matplotlib.pyplot as plt
import palettable
import time
import random

from isothermeos import poly_fit, poly_calc

from matplotlib.axes._axes import _log as matplotlib_axes_logger
matplotlib_axes_logger.setLevel('ERROR')

# Color initialize
cm = palettable.cartocolors.qualitative.Prism_10
color_list = cm.mpl_colors
color_list.insert(0, "black")

pcut = 100 # set of pcut

def readcsv_t0_p0(t0,p0, folderName="data/epaw0"):
    '''
    read the compression curve data, default folder '/data/epaw0'
    return density, pressure, in unit of g/cm^3 and GPa respectively.
    '''

    fileName = "%.1f-%.1f.csv"%(t0,p0)
    df = pd.read_csv(os.path.join(folderName, fileName))
    return b3_to_density(df["v"]), df["p"]

def plot_nif(nif):
    '''
    Draw NIF Data on the current figure
    '''
    plt.errorbar(nif["d"], nif["p"], xerr=nif["d_dev"], yerr=nif["p_dev"],
c=color_list[8], marker=".", mec=color_list[0], ecolor=color_list[6], label="NIF")
```

```
def plot_nif_0(nif):
    '''
    Draw NIF Data on 0
    '''
    plt.errorbar(nif["d"], nif["p"]* 0, xerr=nif["d_dev"], yerr=nif["p_dev"],
                 c=color_list[8], marker=".", mec=color_list[0], ecolor=color_list[6],
label="NIF")

def poly_func(x,a,b,c,d,e):
    return a + b * x + c * x ** 2 + d * x ** 3 + e * x ** 4

def poly_curve_fit(d,p):
    return curve_fit(poly_func, d, p)

def poly_simu(t0,p0):
    '''
    give the initial conditions, t0, po
    return the polynomial 4th order fitting parameters and the density range
    '''
    d,p = readcsv_t0_p0(t0, p0)
    param, _ = poly_curve_fit(d, p)
    lim = (min(d),max(d))
    return param, lim

def deviation(nif, param, lim, pcut, d=[]):
    '''
    calculated dP/P on the nif's densitys, only valid of p > pcut
    return d, pratio
    '''
    if d == []:
        d = nif["d"]

    new_p = poly_func(d,*param)
    p = nif["p"]
    p_dev = nif["p_dev"]
    p_diff = (new_p - p)
    p_ratio = (new_p - p)/p_dev

    # discard outside points
    # new_p_cut = [i for i in new_p if i>= pcut]
    new_p_cut = new_p[-54:]

    return d[-len(new_p_cut):], p_diff[-len(new_p_cut):], p_ratio[-len(new_p_cut):]

def mc_uncertainty(nif,param,lim,pcut,nmc = 5000):
    '''
    '''
```

```
    d_nif = nif["d"]
    p_nif = nif["p"]
    d_dev = nif["d_dev"]
    p_dev = nif["p_dev"]

    p_diff = np.zeros((54, nmc))
    p_ratio = np.zeros((54,nmc))
    for i in range(nmc):
        d_mc = [random.normalvariate(d_nif[j] ,d_dev[j]/2 )  for j in
range(len(d_nif))]
        d_mc = np.array(d_mc)
        d1, pd1, pr1 = deviation(nif, param, lim, pcut, d=d_mc)
        p_diff[:, i] = pd1
        p_ratio[:,i] = pr1

    return d_nif[-len(d1):], np.average(p_diff, axis=1), np.average(p_ratio, axis=1)

def plot_poly_valid(nif, t0, p0, pcut):
    '''
    Figure 0. polynomial fit!
    '''
    param, lim = poly_simu(t0, p0)
    d,p = readcsv_t0_p0(t0, p0)
    pp = poly_func(d,*param)

    fig=plt.figure(figsize=(5,6))
    plt.subplots_adjust(left=0.15, bottom=None, right=0.95, top=None,
                        wspace=None, hspace=0)

    plt.subplot2grid((5, 1), (0, 0), rowspan=4)

    plt.plot(d, p, label="Simulation at $T_0$ = %s, $P_0$ = %s" %
             (t0, p0), lw=2.5, alpha=0.6)
    plt.plot(d, pp, label= "Polynomial 4th order fitting curve")
    plt.legend(frameon=False)
    plt.title("Validation of Polynomial 4th order fitting\nfor EPAW simulation")
    plt.xlim(9.8, 21)
    plt.ylim(0, 1460)
    plt.xlabel("Density (g/cm$^3$)")
    plt.ylabel("Pressure (GPa)")

    left, bottom, width, height = 0.3, 0.55, 0.25, 0.21
    axn = fig.add_axes([left, bottom, width, height])
    plt.plot(d, p, label="Simulation at $T_0$ = %s, $P_0$ = %s" %
             (t0, p0), lw=2.5, alpha=0.6)
    plt.plot(d, pp, label="Polynomial 4th order fitting curve")
```

```
    axn.set_xlabel("Density (g/cm$^3$)")
    axn.set_ylabel("Pressure (GPa)")

    axn.set_xlim([18.6, 18.8])
    axn.set_ylim([1000, 1050])

    plt.subplot2grid((5, 1), (4, 0), rowspan=1)

    plt.plot(d, p - pp, lw=2, alpha = 0.6)
    plt.plot(d, pp - pp)

    plt.xlim(9.8, 21)
    plt.xlabel("Density (g/cm$^3$)")
    plt.ylabel("$\\Delta P$ (GPa)")

    plt.savefig("graph/polyverify.pdf")

def plot_nif_epaw(nif,t0,p0,pcut):
    '''
    Figure 1. NIF with uncertainty and a single EPAW simulation
    '''
    plt.figure(figsize=(5, 7))
    plt.subplots_adjust(left=0.15, bottom=None, right=0.95, top=None,
                        wspace=None, hspace=0)

    plt.subplot2grid((6, 1), (0, 0), rowspan=4)
    plot_nif(nif)

    param,lim = poly_simu(t0, p0)
    x = np.linspace(*lim, 100)
    plt.plot(x,poly_func(x, *param), color=color_list[3], label="Simulation at
$T_0$ = %s, $P_0$ = %s" % (t0, p0),lw=2)

    plt.hlines(pcut,0,25,colors=color_list[5], ls=":",label="cut off pressure")
    plt.legend(frameon=False)
    plt.title("View of NIF data and\n one isentropic EPAW simulation")
    plt.xlim(9.8, 21)
    plt.ylim(0, 1460)
    plt.xlabel("Density (g/cm$^3$)")
    plt.ylabel("Pressure (GPa)")

    dx, p_diff, p_ratio = deviation(nif, param, lim, pcut)

    plt.subplot2grid((6, 1), (4, 0), rowspan=1)
    plot_nif_0(nif)
    plt.plot(dx, p_diff, label="Simulation at $T_0$ = %s, $P_0$ = %s" %
             (t0, p0), lw=2, color=color_list[3])
```

```
    # plt.legend(frameon=False)
    plt.xlim(9.8, 21)
    plt.xlabel("Density (g/cm$^3$)")
    plt.ylabel("$\\Delta P$ (GPa)")

    plt.subplot2grid((6, 1), (5, 0), rowspan=1)
    plt.plot(dx, p_ratio, label="Simulation at $T_0$ = %s, $P_0$ = %s" %
             (t0, p0), lw=2, color=color_list[3])
    # plt.legend(frameon=False)
    plt.xlim(9.8, 21)
    plt.xlabel("Density (g/cm$^3)$")
    plt.ylabel("$\\frac{P_{simu}-P_{NIF}}{\\sigma_{P}}$ (%)")
    plt.savefig("graph/NIF.pdf", dpi=300)

def plot_nif_epaw_mc(nif, t0, p0, pcut):
    '''
    Figure 2. NIF with uncertainty and a single EPAW simulation,
    the bottom figure is using mc_uncertainty
    '''
    plt.figure(figsize=(5, 7))
    plt.subplots_adjust(left=0.15, bottom=None, right=0.95, top=None,
                        wspace=None, hspace=0)

    plt.subplot2grid((6, 1), (0, 0), rowspan=4)
    plot_nif(nif)

    param, lim = poly_simu(t0, p0)
    x = np.linspace(*lim, 100)
    plt.plot(x, poly_func(x, *param),
             color=color_list[3], label="Simulation at $T_0$ = %s, $P_0$ = %s" % (t0,
p0), lw=2)

    plt.hlines(
        pcut, 0, 25, colors=color_list[5], ls=":", label="cut off pressure")
    plt.legend(frameon=False)
    plt.title("View of NIF data and\n one isentropic EPAW simulation")
    plt.xlim(9.8, 21)
    plt.ylim(0, 1460)
    plt.xlabel("Density (g/cm$^3$)")
    plt.ylabel("Pressure (GPa)")

    dx, p_diff, p_ratio = mc_uncertainty(nif, param, lim, pcut)

    plt.subplot2grid((6, 1), (4, 0), rowspan=1)
    plot_nif_0(nif)
    plt.plot(dx, p_diff, label="Simulation at $T_0$ = %s, $P_0$ = %s" %
             (t0, p0), lw=2, color=color_list[3])
```

```
    # plt.legend(frameon=False)
    plt.xlim(9.8, 21)
    plt.xlabel("Density (g/cm$^3$)")
    plt.ylabel("$\\Delta P$ (GPa)")

    plt.subplot2grid((6, 1), (5, 0), rowspan=1)
    plt.plot(dx, p_ratio, label="Simulation at $T_0$ = %s, $P_0$ = %s" %
             (t0, p0), lw=2, color=color_list[3])
    # plt.legend(frameon=False)
    plt.xlim(9.8, 21)
    plt.xlabel("Density (g/cm$^3)$")
    plt.ylabel("$\\frac{P_{simu}-P_{NIF}}{\\sigma_{P}}$ (%)")
    plt.savefig("graph/NIF_mc_1.pdf", dpi=300)

def plot_all_mc(nif,t0_list,p0_list,pcut):

    p_ratio_matrix = np.zeros((len(t0_list), len(p0_list)))
    for i,t0 in enumerate(t0_list):
        for j,p0 in enumerate(p0_list):
            param, lim = poly_simu(t0, p0)
            dx, p_diff, p_ratio = mc_uncertainty(nif, param, lim, pcut,nmc=1000)
            p_ratio_ttl = np.sum(np.abs(p_ratio))
            p_ratio_matrix[i, j] = p_ratio_ttl

    plt.close()
    plt.figure()
    plt.imshow(p_ratio_matrix, cmap="rainbow")
    plt.colorbar()

    init_t_list = np.linspace(500, 2000, 4)
    init_p_list = np.linspace(50, 70, 5)
    yrange = np.linspace(0, 15, 4)
    xrange = np.linspace(0, 20, 5)
    plt.xticks(xrange, init_p_list)
    plt.yticks(yrange, init_t_list)

    plt.title("Deviation map")

    best_fit = np.where(p_ratio_matrix == np.min(p_ratio_matrix))
    print(best_fit)

    plt.scatter(10, 5, marker="*", s=80, color="k")
    plt.scatter(best_fit[1], best_fit[0], marker="*", s=80, color="white")

    plt.xlabel('Initial Pressure (GPa)')
    plt.ylabel('Initial Temperature (K)')
```

```
    plt.savefig("graph/nif_all_1000.pdf")

if __name__=="__main__":

    # Load nif data
    nif = pd.read_csv(os.path.join("data", "nif.csv"))

    # Draw figure 1
    plot_nif_epaw(nif,1000,60,100)

    # Draw figure 2: Do the monte carlo uncertainty test on a single curve
    plot_nif_epaw_mc(nif,1000,60,100)

    # valit of polynomial 4th order fit of epaw!
    plot_poly_valid(nif, 1000, 60, 100)

    # Do the whole thing
    t0_list = np.linspace(500, 2000, 16)
    p0_list = np.linspace(50, 70, 21)
    plot_all_mc(nif,t0_list,p0_list,100)
```