小角散乱データ解析における jupyter notebook の利用

東海国立大学機構・岐阜大学・工学部

藤澤　哲郎

E-mail: fujisawa@gifu-u.ac.jp

要旨

小角散乱のデータ解析は、実験データから 3 次元構造が直接得られるわけではないので、実験データを様々な形式に変換して回帰曲線より構造パラメータを決定する。あるいは、モデルを仮定し、そのパラメータを最適化して実験データにフィットさせることにより構造解析を行うのが一般である。その際、シミュレーション結果を 2 次元ないしは 3 次元でプロットして妥当性を検討する。データを様々な形式に変換してプロットし、回帰分析を行うことは商用のグラフソフト **Igor** などを使えばかなり高度なことも可能である。しかし、有料であり全ての人間の間で解析結果を共有することは難しい。**jupyter-notebook** は python 上で動くフリーの対話型 python 環境で、データサイエンスにおける強力なツールとして急速に普及している。**jupyter-notebook** は python 環境なのでファイルを読み込んで、対数を取ったり、最小自乗近似を行うなどの python 式を実行できるのみならず、数式・テキスト入力、2 次元プロットや構造の 3 次元モデルなども埋め込むことができる。 **jupyter-notebook** では入力、出力情報も記録されるので、 **jupyter-notebook** を読み込むことによって別の PC で内容を確認し、解析の流れを実行することにより追跡できる。計算機環境の違う複数の人間でデータ解析を携わっている場合は非常に便利である。本稿では、2019 年に John Badger 氏によって開発された動径分布関数からタンパク質分子構造を再構成するプログラム **SHAPES** による構造解析を通して、 **jupyter-notebook** 利用に役立つテクニックを紹介する。

なお本稿は、筆者が担当している研究室のホームページに掲載した「小角散乱データ解析における jupyter notebook の利用」をそのまま転載したものである。使用した notebook 等は、

https://www1.gifu-u.ac.jp/~fujilab/jupyter_SAXS_html/download.html

からもダウンロードできる。

1 序論

1.1 jupyter notebook について

python は科学技術計算の分野では最近注目を浴びている言語で、プログラミング初学者が習得しやすい言語の一つとして知られている。フリーで windows、mac、linux など動作する環境が幅広い。その上、数値計算に必要な信頼性の高いライブラリーがそろっており、特に深層学習や機械学習ではアルゴリズムの実装に良く使われている。**jupyter notebook** (https://jupyter.org/) は、対話型 python 環境で、ユーザーがプログラムの断片やデータの一部を入力して、試行錯誤を行い、その結果や振る舞いを眺めながら考える環境を与えることができ、さらにそれらを共有、追体験できるのである。また、 **jupyter notebook** で生成される notebook はネット上に公開し知識を共有するのに適している。 **jupyter notebook** 上での解析工程は一般的な現代データサイエンス解析のワークフローに順じており、その使い方に慣れることは異分野間でのコミュニケーションだけでなく、学生諸君に対しても教育効果は高いと思われる。残念ながら巷であふれている **jupyter notebook** の使い方の情報は csv ファイル形式の汎用データベースに関してが主で、実験データの処理に関して言及されているものは少ない。

本稿ではタンパク質溶液散乱の解析フローを **jupyter-notebook** で実行することにより、タンパク質溶液散乱で良く使われるデータ形式のファイルを **jupyter-notebook** で取り扱うための概要を紹介する。

1.2 実行までの準備

jupyter-notebook を導入するには、最初に python をインストールしてその後に **jupyter-notebook** を使えるようにする。この方法についてはインターネットを探せば、様々な情報が得られる。筆者のおすすめは最も簡単な方法は、**Anaconda** と呼ばれるデータサイエンスに特化した python の無料ディストリビューションの一部として **jupyter-notebook** をインストールするのが便利である。

Anaconda は、以下のサイトからダウンロードすることができる。

https://www.anaconda.com/download

windows、 mac、 linux 向けにインストーラーが準備されている。インストールする際には python のバージョンを選択しないといけないが、サポートが終了した 2.7 よりは 3.x を選択するほうが良いであろう。

元々、 **Anaconda** はデータサイエンスに特化したプラットフォームを提供したいというのが目的なので、それ用によく使われる一連の python パッケージ群が同梱されている。

jupyter-notebook を起動するにはターミナルを開きコマンドプロンプトで:

```
jupyter-notebook
```

と打てば、勝手にブラウザーが立ち上がる。

デフォルトでは、3 次元構造の pdb フォーマットを取り扱えないので、**nglview** [1] という widget をインストールする。

https://github.com/arose/nglview

コマンドプロンプトで:

```
conda install nglview -c conda-forge
jupyter-nbextension enable nglview --py --sys-prefix
```

でエラーがでなければ大丈夫である。グラフィックボード環境の違いにより表示されたりされなかったりすることもあるが、この widget を使えば、3 次元構造を **jupyter-notebook** 上でぐるぐる回すことができるようになる。

また、本稿では EMBL Hamburg の小角散乱グループが開発したタンパク質溶液散乱の標準的パッケージ **ATSAS** を使用する [2] 。以下のサイトでユーザー登録すると無料でダウンロードできるのであらかじめインストールしておく。

https://www.embl-hamburg.de/biosaxs/download.html

1.3 本稿で取り扱うデータ解析の流れ

本稿では **jupyter-notebook** を用いてタンパク質溶液散乱データの読み方、プロットの仕方などを初心者がつまづきやすい点を中心に紹介していきたい。 タンパク質溶液散乱では、最も一般的な解析は散乱データからタンパク質構造を充填球で近似したいわゆるビーズモデルを計算することである。 **ATSAS** で使用される代表的なフォーマットのファイルを扱うよう、以下のデータ解析を **jupyter-notebook** で処理してみたい。

1. タンパク質結晶構造 (`*.pdb` フォーマット) から散乱曲線 (`*.dat` フォーマット) を計算しプロットする。
2. 散乱曲線 $I(q)$ から動径分布関数 $P(r)$ 関数を計算し、出力ファイル（ `*.out` フォーマット）から $P(r)$ 関数を抽出しプロットする。

3. $P(r)$ 関数からビーズモデルを計算する。シミュレーションデータ（ `*.fit` フォーマット）と元データを比較する。
4. ビーズモデル（ `*.pdb` ）と結晶構造（ `*.pdb` ）の比較

解析フローは 図 1 となる。

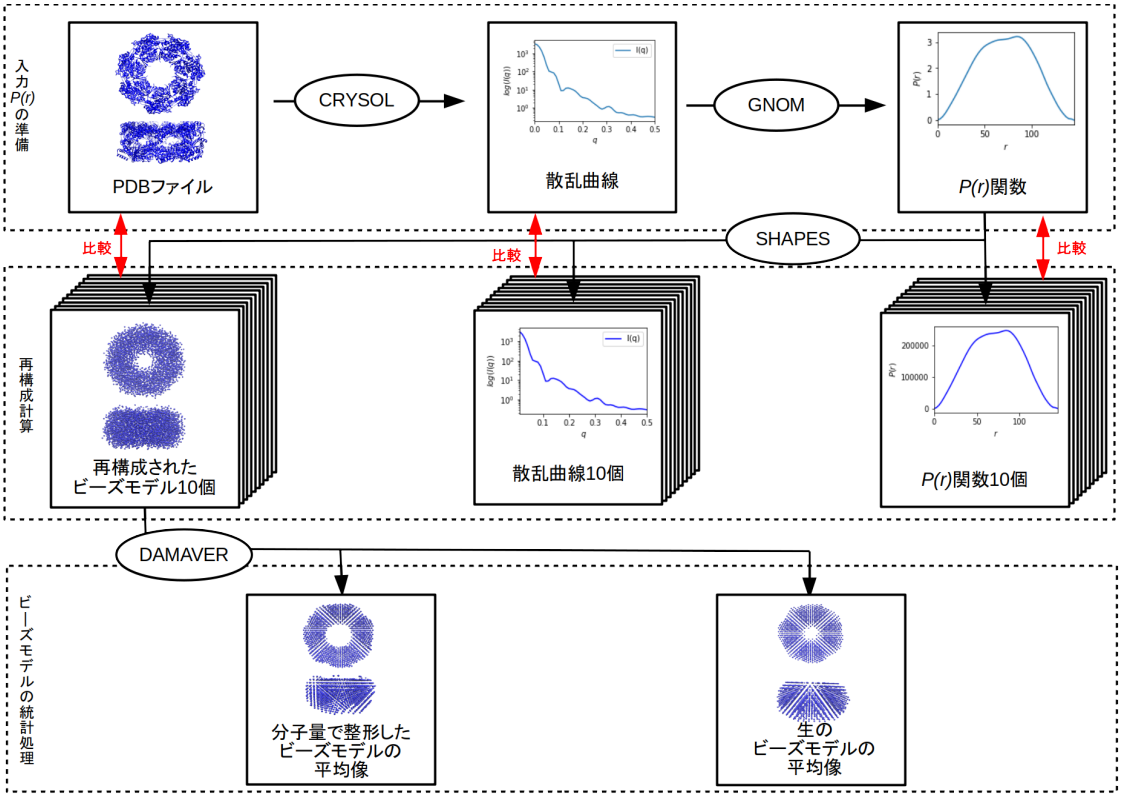


図 1: 本稿での解析フロー

使用するデータ：

- タンパク質結晶構造 `6lyz.pdb`
- タンパク質溶液散乱データ `lyzexp.dat` [3]

1.4 動径分布関数からビーズモデルを計算するプログラム SHAPES

小角散乱データからビーズモデルを計算するのは一般的に **DAMMIN** というプログラムが広く使用されている。これは $I(q)$ に対してフィッテイングを行いビーズモデルを再構成計算する。本稿では、2019 年に発表された **SHAPES** [4] による構造計算を行ってみよう。

SHAPES は John Badger によって開発された $P(r)$ 関数に対してフィッティングを行い、タンパク質分子構造を再構成するプログラムである。元の python コードは以下のサイトからフリーでダウンロードできる。

http://saxs2shapes.com

元々 **SHAPES** は単体で python コードを実行することを想定しているが、**jupyter-notebook** 環境では使いにくいので変更を加えた。具体的には変数はセル上に入力して **SHAPES** をセル上から実行できるように **SHAPES** を本体部 **shapes_main** とモジュール部 **shapes_module** に分割した。

shapes_module は小角散乱を取り扱う上で便利なメソッドを含有している。 **SHAPES** 実行以外でも色々活用できる。その詳細を以下に示す。

表 1: **shapes_module**

メソッド名	内容
ft_to_intensity	動径分布関数 $P(r)$ 関数から散乱強度 $I(q)$ の計算
score_Ic	入力散乱強度 $I(q)$ と出力散乱強度 $I_{calc}(q)$ のスケーリングと比較
write_all_data	入力データと出力データを書き込む
read_i	**GNOM** 出力データから散乱強度データを読み取る
read_pdb	初期 PDB ファイルを読み取る
pr_writer	計算された $P(r)$ データをファイルに書き込む
pdb_writer	ビーズの位置情報を PDB ファイル形式で書き込む
set_box	概念的な格子上のローカルビーズ密度と点密度を評価する
set_vol	分子体積の設定
disallowed_beads	許容体積内にないビーズを見つける
calc_pr	$P(r)$ 関数の計算
pr_dif	入力 $P(r)$ とモデル $P(r)$ の rms 差のスコア
pr_rfactor	入力 $P(r)$ とモデル $P(r)$ の残差の統計
vdw_energy	ビーズ間の相互作用エネルギーを計算する
random_beads	最大長 D_{max} の格子にビーズをランダムに配置する
read_pr	**GNOM** 出力ファイルから $P(r)$ 関数を抽出する
scale_pr	入力 $P(r)$ 関数と出力 $P(r)$ をスケーリング
get_dr	2 つの座標間の 0 以外の距離を返す
center_beads	半径内のビーズの中心を見つける
get_total_energy	総 vdw エネルギーの平均を取得
e_min	エネルギーを最小化する
make_symm	対称性に関係するパラメータを設定する
pr_shift_atom	ビーズの位置を変更するための $P(r)$ のシフトベクトルを設定する
recenter_pdb	元のビーズセットを原点に移動

SHAPES のアルゴリズムは以下のようである。

このプログラムはビーズ間のポテンシャル関数として変形レナード・ジョーンズポテンシャルを用いて、ビーズの位置の最適化を試みる。

SHAPES の入力パラメータ

SHAPES で使用されるパラメータとその内容を以下に示す。

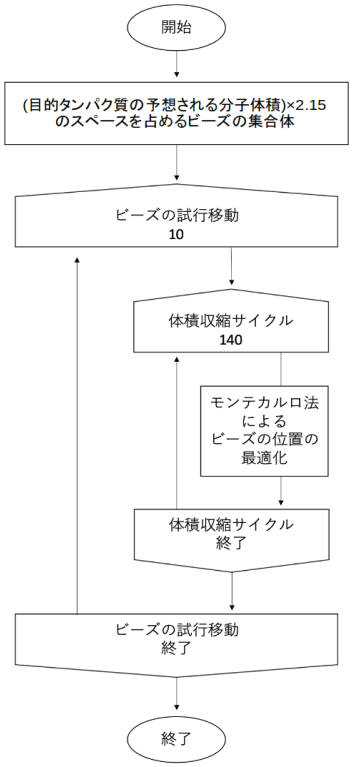


図 2: **SHAPES** のアルゴリズム

表 2: **SHAPES_main** の入力パラメータ

パラメータ名	内容	デフォルト
nbeads_h	構造内のアミノ酸の総数	なし・入力必須
inFile_h	ATSAS パッケージの **GNOM** による出力としての $P(r)$ ファイル	なし・入力必須
num_sols	再構成の実行回数	4 回
num_aa	計算に使用されるビーズの数を変更するために使用するアミノ酸数のスケール	1.0
num_symm	タンパク質集合体の回転対称性	1
bias_z_h	Z 軸に沿ってビーズの初期構成を偏らせる	0.0
inflate	予想される部分比容に関連して、開始体積と最終体積を調整するためのスケール係数	1.0
pdbfile_in_h	入力 PDB ファイルから読み取られた Cα 原子を使用して、再構成プロセスを開始する	none
surface_scale	再構成手順を安定させ、より連続した体積を持ち互いに類似した再構成をもたらす	0.0

分子体積 [Å^3] は入力されたアミノ酸数から推定される。アミノ酸あたり 110Da を想定し、分子量と分子体積の間の変換係数として 1.21 を使用する。

.

2 データ解析の実際

2.1 結晶構造より理論散乱曲線の計算

結晶構造より理論散乱曲線を計算するのに使用されるプログラムとして **crysol** がある [3]。このソフトはターミナルからコマンドを打つことにより実行されるが、 **jupyter-notebook** のセルからシェル・コマンドを実行させるには、 ！ 記号を先頭につけてコマンドを打てば良い。 !! としているところはシェル・コマンドの出力結果を配列化しているだけである。

crysol の実行

crysol のヘルプを表示させてみると、

[ ]:
```
!crysol -h
```

[ ]:
```
  Usage in the batch mode:
  crysol [<Inp_File1>]...[<Inp_FileK>] [<Dat_File>]
  [-param1 <param1>]...[-paramN <paramN>]

    :
  途中略
    :

  Examples:

  crysol 6lyz.pdb -lm 20
  Calculate scattering intensity from the PDB file
  6lyz.pdb with Lmax = 20 and without fitting

  crysol mod*.pdb exp.dat -un 2
  Process PDB files with the names beginning with
  "mod" and fit experimental data exp.dat with
  scattering vector given in inverse nanometres.

  crysol *.sav lyzexp.dat
  Restore the scattering intensity from all
  sav files in the current directory and fit
  the experimental data in the file lyzexp.dat

Report bugs to <atsas@embl-hamburg.de>.
```

ご覧のように、コマンド実行の結果がセルとして表示されているのがわかる。

今、リゾチウムの結晶構造から `6lyz.pdb` を計算してみよう。 `crysol` の引数はデータ点数を 201 とするために `-ns 201` としている。

[ ]:
```
!crysol 6lyz.pdb -lm 20 -ns 201
```

[ ]:
```
***  ------------------------------------------------  ***
  ***    C R Y S O L  Wintel/UNIX/Linux version 2.8.4    ***
  ***  Please reference: D.Svergun, C.Barberato          ***
  ***   & M.H.J.Koch (1995) J. Appl.Cryst., 28, 768-773  ***
  ***   Version (LMAX=99) for small and wide angles      ***
  ***        Last revised  ---  10/12/18 10:00           ***
  ***   Copyright (c) ATSAS Team                         ***
  ***   EMBL, Hamburg Outstation, 1995 - 2018            ***
  ***  ------------------------------------------------  ***

  ------------------------------------------------
                Program options :
  0 - evaluate scattering amplitudes and envelope
  1 - evaluate only envelope and Flms
  2 - read CRYSOL information from a .sav file
   Type crysol -help for batch mode use
  ------------------------------------------------

  ------------------------------------------------
          Following file names will be used:
 6lyz01.log -- CRYSOL log-file          (ASCII)
 6lyz01.int -- scattering intensities   (ASCII)
 6lyz01.alm -- net partial amplitudes   (binary)
  ------------------------------------------------
```

(次のページに続く)

(前のページからの続き)

```
 ----------  Reciprocal space grid  -------------
   ( in s = 4*pi*sin(theta)/lambda [1/angstrom] )
Read atoms and evaluate geometrical center ...
 Number of atoms read ................................ : 1001
 Number of discarded waters .......................... : 101
Percent processed      10  20  30  40  50  60  70  80  90 100
Processing atoms   :>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
Center of the excess electron density:  0.452 -0.004  0.276
Processing envelope:>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
Processing envelope:>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
--- Structural parameters (sizes in angstroms) ---
Electron    rg    :  13.99       Envelope   Rg       :  14.08
Shape       Rg    :  13.97       Envelope   volume   : 0.1806E+05
Shell     volume  : 0.1129E+05   Envelope   surface  :  3150.
Shell       Rg    :  18.98       Envelope   radius   :  25.49
Shell     width   :  3.000       Envelope   diameter :  49.04
Molecular weight: 0.1432E+05     Dry volume          : 0.1736E+05
Displaced volume: 0.1741E+05     Average atomic rad.:  1.607
Number of residuals :   129
Average atomic volume ............................... : 17.40
Rg ( Atoms - Excluded volume + Shell ) .............. : 15.04
Rg from the slope of net intensity .................. : 15.41
Average electron density ............................ : 0.4383
Intensities     saved to file 6lyz01.int
I_abs(s)[cm^-1]/c[mg/ml] saved to file 6lyz01.abs
Net amplitudes saved to file 6lyz01.alm
```

abs、alm、int それにプログラムのログである log　という拡張子のファイルが crysol 実行毎に生成される。

散乱曲線のデータは abs と int であるが使うのは int の方である。

実際にファイルの中身を一部表示させると、

[ ]:
```
!head "6lyz00.int"
```

[ ]:
```
 Dif/Atom/Shape/Bord 6lyz.pdb     Dro:  0.030 Ra: 1.6074 Rg: 15.41 Vtot:  18057.
  0.000000E+00  0.484317E+07  0.582674E+08  0.338289E+08  0.147208E+06
  0.250000E-02  0.484078E+07  0.582433E+08  0.338151E+08  0.147088E+06
  0.500000E-02  0.483365E+07  0.581719E+08  0.337741E+08  0.146731E+06
  0.750000E-02  0.482179E+07  0.580530E+08  0.337059E+08  0.146137E+06
  0.100000E-01  0.480523E+07  0.578869E+08  0.336106E+08  0.145308E+06
  0.125000E-01  0.478402E+07  0.576741E+08  0.334884E+08  0.144250E+06
  0.150000E-01  0.475822E+07  0.574149E+08  0.333397E+08  0.142965E+06
  0.175000E-01  0.472792E+07  0.571101E+08  0.331648E+08  0.141462E+06
  0.200000E-01  0.469315E+07  0.567603E+08  0.329640E+08  0.139740E+06
```

ここで最初の行は理論散乱曲線を計算するためのパラメータで、使用するのは 1 列目の q と 2 列目の $I(q)$ のみである。残念ながら crysol の出力はすぐ使えるようになっていないので加工して保存してやる必要がある。

理論データの保存

さっそくこのファイルを読み込んでみよう。普通に python でファイルを読むこともできるが、せっかくなので、 pandas (https://pandas.pydata.org/) と呼ばれる python で データ解析を行うためのパッケージを使ってみる。

pandas には 1 次元の Series と 2 次元の DataFrame という 2 つのデータ構造が用意されているが、小角では普通 DataFrame を使う。

pandas だけでなく描画や数値計算など必要なモジュールをインポートしておく

[1]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

タンパク質溶液散乱の場合、散乱強度 $I(q)$ は重量濃度 c で割った形で実験値として使用するので、 2 列目の $I(q)$ を分子体積 Vtot で割っておく。

Vtot は header に書きこまれているので、最初に Vtot だけ取り出しておく。

[2]:
```
f=open("6lyz00.int")
header=f.readline()
Vtot=float(header.split(":")[-1])
print('Vtot=',Vtot)
```

```
Vtot= 18057.0
```

[3]:
```
df=pd.read_csv("6lyz00.int",delim_whitespace=True,header=0,names=["q","I"],usecols=[0,1])
```

[4]:
```
df
```

[4]:
```
          q          I
0    0.0000  4843170.0
1    0.0025  4840780.0
2    0.0050  4833650.0
3    0.0075  4821790.0
4    0.0100  4805230.0
..      ...        ...
196  0.4900    24998.3
197  0.4925    24965.8
198  0.4950    24943.6
199  0.4975    24931.3
200  0.5000    24928.1

[201 rows x 2 columns]
```

先程書いたように $I(q)$ を $I(q)/c$ に対応させるために 2 列めに $I(q)/Vtot$、3 列目には各点の誤差を入れるのが小角散乱における標準フォーマット なので仮想的に $I(q)$ の 3%の値を入れておく。

[5]:
```
df["I"]=df["I"]/Vtot
df["sI"]=df["I"]*3./100
df
```

[5]:
```
          q           I        sI
0    0.0000  268.215650  8.046470
1    0.0025  268.083292  8.042499
2    0.0050  267.688431  8.030653
3    0.0075  267.031622  8.010949
4    0.0100  266.114526  7.983436
..      ...         ...       ...
196  0.4900    1.384410  0.041532
197  0.4925    1.382611  0.041478
198  0.4950    1.381381  0.041441
199  0.4975    1.380700  0.041421
200  0.5000    1.380523  0.041416

[201 rows x 3 columns]
```

このようにして作った df をファイル 6lyz.dat に出力する。 header 行だけ先に書いておき

[6]:
```
f=open("6lyz.dat","w")
headerline="# q     I(q)      sI(q)\n"
f.write(headerline)
f.close()
```

次に data を追記する。フォーマットは指定できる。

[7]:
```
df.to_csv("6lyz.dat",sep=' ',mode="a",float_format='%.6e', header=False, index=False)
```

ファイルの中身は

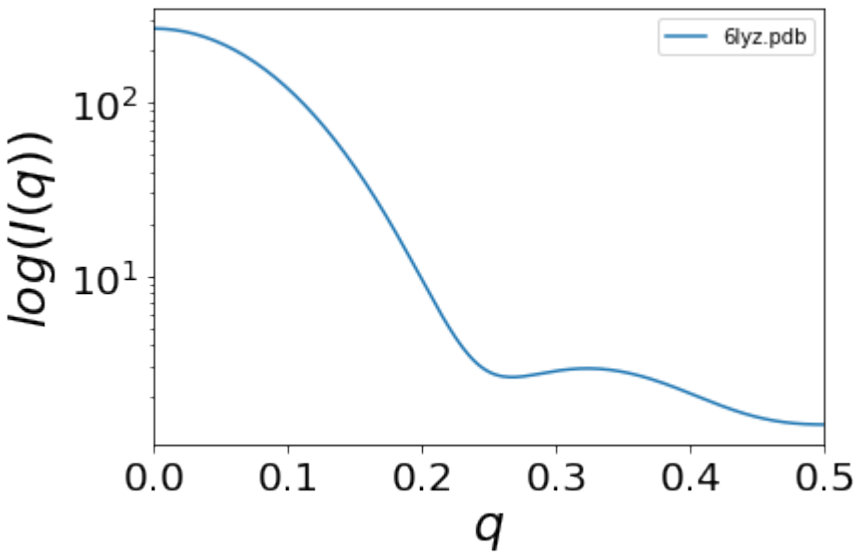
[ ]:
```
!head "6lyz.dat"
```

[ ]:
```
# q     I(q)      sI(q)
0.000000e+00 2.682157e+02 8.046470e+00
2.500000e-03 2.680833e+02 8.042499e+00
5.000000e-03 2.676884e+02 8.030653e+00
7.500000e-03 2.670316e+02 8.010949e+00
1.000000e-02 2.661145e+02 7.983436e+00
1.250000e-02 2.649399e+02 7.948197e+00
1.500000e-02 2.635111e+02 7.905333e+00
1.750000e-02 2.618331e+02 7.854993e+00
2.000000e-02 2.599075e+02 7.797225e+00
```

データのプロット

データのプロットは python の matplotlib (https://matplotlib.org/) を使うのが一般的である。 縦軸に $I(q)$ の対数を取る。

[9]:
```
ax=df.plot(x='q',y='I',logy=True,fontsize=20,label='6lyz.pdb')
ax.set_xlabel('$q$',fontsize=25)
ax.set_ylabel('$log(I(q))$',fontsize=25)
#plt.savefig('Iq.png')
```



[ ]:

このように **crysol** の実行結果を次の作業工程にのせようとすると一手間が必要となる。

2.2 理論散乱曲線から動径分布関数の計算

動径分布関数 $P(r)$ は 2 点間距離分布関数とも呼ばれ、タンパク質内の原子の 2 点間距離のヒストグラムを表す関数である。これは、散乱曲線 $I(q)$ （ $q = 4\pi \sin\theta/\lambda$, 2θ ：散乱角、 λ :波長）から

$$P(r) = \frac{1}{2\pi^2}\int_0^\infty qrI(q)\sin(qr)dq$$

の式で得ることができる。しかし、直接上式を計算すると窓関数がエイリアシングにより $P(r)$ 関数に乗ってしまうので、実務では間接フーリエ変換を用いる。

間接フーリエ変換では、 $P(r)$ 関数を基底関数系 $\varphi_i(r)$ を用いて以下のように近似する。

$$P_{fit}(r) = \sum_{i=1}^{n} c_i\varphi_i(r)$$

$I_{fit}(q)$ は $P_{fit}(r)$ をフーリエ変換したものだから、

$\mathcal{F}[\varphi_i(r)] = \psi_i(q)$ とすると、フーリエ変換の線形性により、

$$I_{fit}(q) = \sum_{i=1}^{n} c_i\psi_i(q)$$

となる。間接フーリエ変換とは $I(q)$ に対し $I_{fit}(q)$ による最小二乗回帰で c_i を求め間接的に $P_{fit}(r)$ を求める。

小角散乱データ処理用の間接フーリエ変換プログラムの代表的なものに **gnom** [5] がある。このプログラムの出力も取り扱いにくい。

gnom のヘルプを見てみると、

[ ]:
```
!gnom -h
```

[ ]:
```
# Usage: gnom [OPTIONS] <FILE>

Indirect transform for SAS data processing -- evaluates the P(r)

Known Arguments:
  FILE                        Experimental data file

Known Options:
  -h, --help                  Print usage information and exit
  -v, --version               Print version information and exit
      --seed=<INT>            Set the seed for the random number generator
      --first=<N>             first point of the data file to use (default: 1)
      --last=<N>              last point of the data file to use (default: all)
      --system=<N>            system type, one of 0...6 (default: 0)
      --rmin=<VALUE>          minimum characteristic size of SYSTEM (default: 0.0)
      --rmax=<VALUE>           maximum characteristic size of SYSTEM (required)
      --rad56=<VALUE>         (no description)
      --force-zero-rmin=<Y|N>Zero condition at r=rmin (default: YES)
      --force-zero-rmax=<Y|N>Zero condition at r=rmax (default: YES)
      --nr=<N>                number of points in real space (default: automatic)
      --alpha=<VALUE>         alpha value (default: automatic)
  -o, --output=<FILE>         output file name (default: stdout)

Mandatory arguments to long options are mandatory for short options too.

Report bugs to <atsas@embl-hamburg.de>.
```

gnom に入力ファイルと最大長 $Dmax$ の予想値を入れてやると $P_{fit}(r)$ 関数が出力される。

[ ]:
```
!gnom 6lyz.dat -rmax 50 -o 6lyz.out
```

[ ]:
```
!head -n +60 "6lyz.out"
```

[ ]:
```
           ####      G N O M             Version 5.0 (r12314)      ####
                                               Fri Apr 10 11:30:41 2020

           ####      Configuration                                 ####

            :
           略
            :

           ####      Results                                       ####

    Parameter    DISCRP  OSCILL  STABIL  SYSDEV  POSITV  VALCEN  SMOOTH
    Weight        1.000   3.000   3.000   3.000   1.000   1.000   1.000
    Sigma         0.300   0.600   0.120   0.120   0.120   0.120   0.600
    Ideal         0.700   1.100   0.000   1.000   1.000   0.950   0.000
    Current       0.003   1.349   0.002   0.100   1.000   0.947   0.103
           ------------------------------------------------------------
    Estimate      0.005   0.842   1.000   0.000   1.000   0.999   0.971

    Angular range:                      0.0000 to       0.5000
    Reciprocal space Rg:             0.1536E+02
    Reciprocal space I(0):           0.2682E+03

    Real space range:                   0.0000 to      50.0000
    Real space Rg:                   0.1535E+02 +-   0.1278E+00
    Real space I(0):                 0.2682E+03 +-   0.1894E+01

    Highest ALPHA (theor):           0.4031E+06
    Current ALPHA:                   0.5615E-02

    Total Estimate:                      0.6538 (a REASONABLE solution)



           ####      Experimental Data and Fit                     ####

      S          J EXP       ERROR       J REG       I REG

   0.000000E+00   0.268216E+03   0.804647E+01   0.268197E+03   0.268197E+03
   0.250000E-02   0.268083E+03   0.804250E+01   0.268066E+03   0.268066E+03
```

(次のページに続く)

(前のページからの続き)

```
  0.500000E-02   0.267688E+03   0.803065E+01   0.267671E+03   0.267671E+03
                        :
                       略
                        :
  0.492500E+00   0.138261E+01   0.414783E-01   0.138267E+01   0.138267E+01
  0.495000E+00   0.138138E+01   0.414414E-01   0.138123E+01   0.138123E+01
  0.497500E+00   0.138070E+01   0.414210E-01   0.138026E+01   0.138026E+01
  0.500000E+00   0.138052E+01   0.414157E-01   0.137969E+01   0.137969E+01

          ####      Real Space Data                                ####

          Distance distribution  function of particle


     R           P(R)       ERROR

 0.0000E+00  0.0000E+00  0.0000E+00
 0.4464E+00  0.1602E-01  0.1713E-01
                :
               略
                :
 0.1607E+02  0.9318E+00  0.4651E-01
 0.1652E+02  0.9464E+00  0.5132E-01
```

上に表示されているのは、出力ファイル `6lyz.out` の中身の一部である。

これからもわかるように、ファイルは単純な構造をしておらず、最初にフィットパラメータ、次に散乱曲線とのフィット、最後にお目当ての $P(r)$ 関数が入っている。

必要な部分を gnom の output から抽出する python ルーチンは、先程紹介した `shapes_module` の `read_pr` というメソッドを使用してもよいが、ここでは以下の筆者のコード `out_read` を紹介する。

[1]:
```
def out_read(fn):
# input: file name (str)
# output: q,I_{exp}(q),sI_{exp}(q),I_{fit}(q),r,P(r),sP(r),Rg from P(r),I(0) from P(r)
# written by T.Fujisawa
#
    fin=open(fn,'r')
    lines = []
    for line in fin:
        #print line
        lines.append(line)
    npts=len(lines)
    j=0
    while j < npts:
        linet=lines[j]
        if 'Real space Rg:' in linet:
            a=linet.split(':')[-1]
            Rg_r=float(a.split('+-')[0])
            j=j+1
        elif 'Real space I(0):' in linet:
            a=linet.split(':')[-1]
            I0_r=float(a.split('+-')[0])
            j=j+1
        elif '      S          J EXP       ERROR       J REG       I REG' in linet:
            iqcash=[]
            j=j+2
            linet=lines[j]
            while 'Distance' not in linet:
                iqcash.append(linet)
                j=j+1
                linet=lines[j]
            j=j+1
            linet=lines[j]
        elif '  R           P(R)      ERROR' in linet:
            prcash=[]
            j=j+1
            while j < npts-1:
                j=j+1
                linet=lines[j]
                prcash.append(linet)
        else:
            j=j+1

    ra=[]
    Pra=[]
    sPra=[]

    for line in prcash:
        rt=line[:14]
        Prt=line[15:26]
        sPrt=line[27:38]
        #if ('    ' not in iqt)or(qt!='n'):
        if rt.isspace():
```

(次のページに続く)

(前のページからの続き)

```
            break
        if '#' in rt:
            continue
        elif '    ' not in Prt:
            ra.append(float(rt))
            Pra.append(float(Prt))
            sPra.append(float(sPrt))


    qe=[]
    iqe=[]
    siqe=[]
    iqm=[]
    for line in iqcash:
        qt=line[:17]
        iqt=line[18:32]
        siqt=line[33:47]
        iqmt=line[48:62]
        #if ('    ' not in iqt)or(qt!='n'):
        if qt.isspace():
            break
        if '#' in qt:
            continue
        elif '    ' not in iqt:
            #print iqt
            qe.append(float(qt))
            iqe.append(float(iqt))
            siqe.append(float(siqt))
            iqm.append(float(iqmt))


    return (qe,iqe,siqe,iqm,ra,Pra,sPra,Rg_r,I0_r)
```

上の関数 out_read に out ファイルを入れてやると目的の情報を抽出してくれる。

[2]:
```
q,I,sI,Im,r,Pr,sPr,Rg,I0=out_read('6lyz.out')
```

基本的に jupyter-notebook においては入力ファイルを notebook ファイルと同じディレクトリに入れておくものだが、ファイル名の打ち間違いなども あるので筆者は GUI で処理するようにしている。python においてはデフォルトの GUI、thkinter を使用する。入力したファイル名はセルに出力するようにしておく。

上の実行例は以下のように変更すると使いやすくなる。

[3]:
```
import tkinter
from tkinter import messagebox as tkMessageBox
from tkinter import filedialog as tkFileDialog
from tkinter import simpledialog as tkSimpleDialog

root=tkinter.Tk()
root.withdraw()

inFile=tkFileDialog.askopenfilename(title='Open GNOM file',filetypes=[("I(q) file","*.out")])
print("GNOM file:",inFile)
q,I,sI,Im,r,Pr,sPr,Rg,I0=out_read(inFile)
```

```
GNOM file: /media/fujisawa/2124879b-e9d4-4135-8fd5-9b114fd41989/home/
↪backup/gifu_19/grad/SAXS_WG/report/source/M_M/6lyz.out
```

これで、q,I,sI,Im,r,Pr,sPr,Rg,I0　の情報が格納された。

crysol の時と同様、pandas にデータを格納してプロットをする。

必要なモジュールをインポートしておき、

[4]:
```
%matplotlib inline
import pandas as pd
import shapes_module as sm
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize
```

まずは散乱曲線のフィットを df というデータフレームに変換する

[5]:
```
df = pd.DataFrame({'q': q, 'I(q)': I,'error':sI,'I(q)_fit':Im})
df.head()
```

[5]:
```
        q     I(q)    error  I(q)_fit
0  0.0000  268.216  8.04647   268.197
1  0.0025  268.083  8.04250   268.066
2  0.0050  267.688  8.03065   267.671
3  0.0075  267.032  8.01095   267.014
4  0.0100  266.115  7.98344   266.098
```

同様に $P(r)$ 関数の方も df2 というデータフレームに変換する。

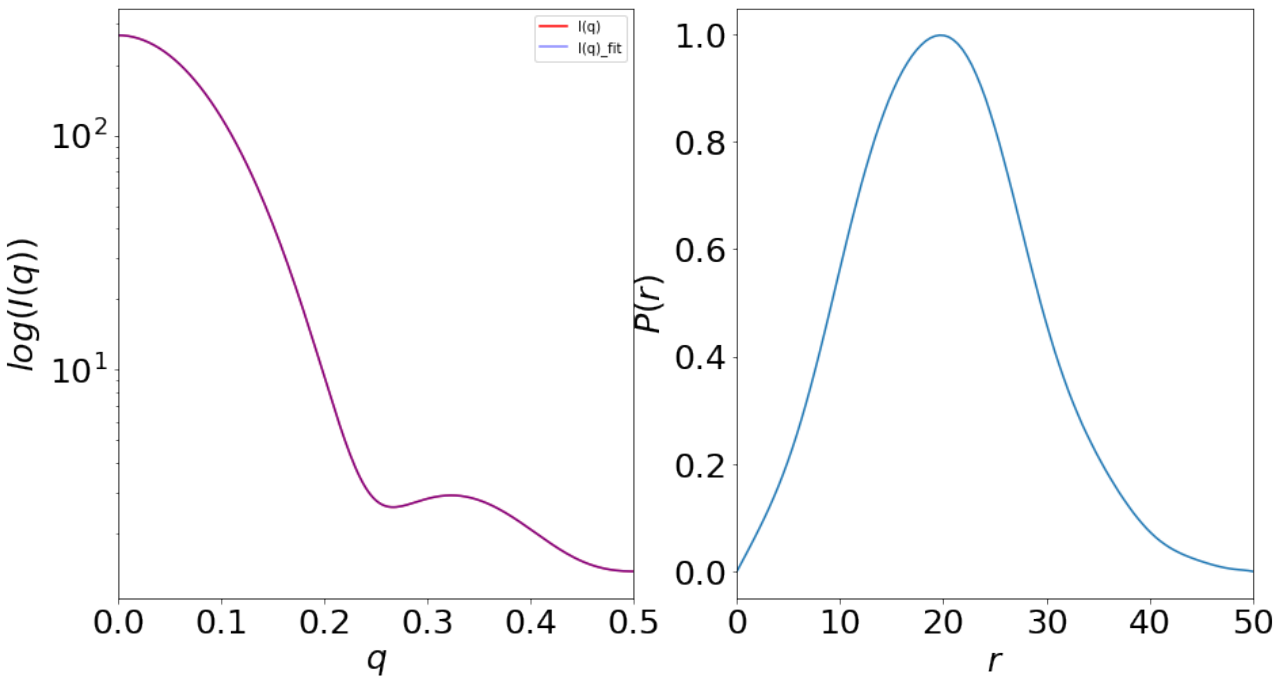
[6]:
```
df2 = pd.DataFrame({'r': r, 'P(r)': Pr,'Error':sPr})
df2.head()
```

[6]:
```
        r     P(r)    Error
0  0.0000  0.00000  0.00000
1  0.4464  0.01602  0.01713
2  0.8929  0.03216  0.02797
3  1.3390  0.04850  0.03289
4  1.7860  0.06513  0.03254
```

間接フーリエ変換の結果のチェックは $I(q)$ のフィットの具合と $P(r)$ の形などで判断するのでプロットをすると、

[7]:
```
fig = plt.figure(figsize=(15, 8))

ax1 = fig.add_subplot(121)
df.plot(ax=ax1,x='q',y='I(q)',color='red',label='I(q)',fontsize=25,logy=True)
df.plot(ax=ax1,x='q',y='I(q)_fit',color='blue',alpha=0.5,label='I(q)_fit',fontsize=25,logy=True)
ax1.set_xlabel('$q$',fontsize=25)
ax1.set_ylabel('$log(I(q))$',fontsize=25)

ax2 = fig.add_subplot(122)
df2.plot(ax=ax2, legend=False,x='r',y='P(r)',fontsize=25)
ax2.set_xlabel('$r$',fontsize=25)
ax2.set_ylabel('$P(r)$',fontsize=25)

#fig.savefig("gnom_plot.png")
```

[7]:
```
Text(0, 0.5, '$P(r)$')
```



2.3 動径分布関数からビーズ構造の計算

SHAPES の実行

まずは必要なモジュールをインポートする

[1]:
```
import shapes_main as shp
import tkinter
from tkinter import messagebox as tkMessageBox
from tkinter import filedialog as tkFileDialog
from tkinter import simpledialog as tkSimpleDialog
```

実際に SHAPES を実行してみる。

shapes_main.py と shapes_module.py を同一ディレクトリにいれておき、shapes_main.py を call する。

入力：num_symm、num_aa,num_sols、nbeads、inFile(GNOM アウトファイル)

出力：ビーズモデル (pdb ファイル)、散乱強度 (dat ファイル)、動径分布関数 (dat ファイル)、偏比容ビーズモデル (pdb ファイル)

出力はビーズモデルだけでなく、ビーズモデルをタンパク質の体積で整形した偏比容ビーズモデルがある。 プログラムは、あらかじめ出力を格納する

[ ]:
```
root=tkinter.Tk()
root.withdraw()

### default parameters ###
aList_summary=[]
bias_z=0.0
inflate=1.0

surface_scale=0.0
starting_pdb='no'
pdbfile_in='none'
#########################

prefix='6lyz_'
num_symm=1 #対称性
num_aa=1.0 #スケール係数
num_sols=10 #試行回数
nbeads=129 #アミノ酸数
inFile=tkFileDialog.askopenfilename(title='Open GNOM file',filetypes=[("アウトファイル","*.out")])
print("GNOM file:",inFile)
shp.main(aList_summary,nbeads,num_sols,num_aa,num_symm,bias_z,inflate,prefix,surface_scale,starting_
 ↪pdb,inFile,pdbfile_in)
```

[ ]:
```
GNOM file: /media/fujisawa/2124879b-e9d4-4135-8fd5-9b114fd41989/home/backup/gifu_19/grad/SAXS_WG/
 ↪report/source/M_M/6lyz.out
Program: SHAPES version 1.3
Author: John Badger
Copyright: 2019, John Badger
License: GNU GPLv3
Number of runs: 1
Number of amino acids: 129
Input P(r) file name: /media/fujisawa/2124879b-e9d4-4135-8fd5-9b114fd41989/home/backup/gifu_19/grad/
 ↪SAXS_WG/report/source/M_M/6lyz.out
Scale aa to bead count: 1.0
Point symmetry: 1
Z-axis bias: 0.0
PSV inflation factor: 1.0
Number of points read from P(r): 113
Grid sampling: 0.4464 Dmax: 50.0
Number of intensity data points read: 200

Reconstruction trial: 1
Number of beads randomly placed: 129
Minimize energy of initial positions
Emin cycle: 0 Energy: 89797276708.35
Emin cycle: 10 Energy: 997.36
Emin cycle: 20 Energy: 233.82
Initial rms P(r): 0.453
Target volume: 2.30 Actual volume: 2.14 Beads outside volume: 0
                :
               略
                :
Target volume: 1.15 Actual volume: 1.14 Beads outside volume: 2

Final model statistics
Delta P(r): 0.093
VDW energy: -1.34
Final PSV of protein envelope: 1.08
Rvalue: 0.021 CHI-squared: 221.934
Output intensity file: 6lyz_intensity_1.dat

Completion time: Fri Apr 10 17:13:01 2020

            これが 10 回続く
```

実行の結果は `prefix` を `6lyz_` としたので

- 6lyz_intensity_#.dat： $I_{fit}(q)$
- 6lyz_pr_calc_#.dat： $P(r)$
- 6lyz_beads_#.dat: ビーズモデル
- 6lyz_psv_shape_#.dat：偏比容ビーズモデル

が生成される。#は番号を表す。

SHAPES 実行結果のチェック

元々 SHAPES は、$P(r)$ 関数に対してシミュレーションを行っているので、動径分布関数のチェックだけでも良いが念の為、散乱強度においてもシミュレーションのチェックを行う。

・散乱曲線の比較

Original data と Output data

・動径分布関数の比較

Original data と Output data

の 2 点に関して、ビーズモデルから予想される $P(r)$、$I_{fit}(q)$ を比較する。

両者の表示とも、結果ファイルを一括して選択した後、各ファイルのラベルを入力するプログラムになっている。

始めに、ファイルの構造をチェックしておこう。

$I_{fit}(q)$ に関しては

[1]:
```
!!head 6lyz_intensity_1.dat
```

[1]:
```
['# Rvalue: 0.012 CHI-squared: 85.097',
 '0.0025 268.083 268.31071303293044',
 '0.005 267.688 267.91908469451465',
 '0.0075 267.032 267.2675296096724',
 '0.01 266.115 266.3577811119744',
 '0.0125 264.94 265.19225612372185',
 '0.015 263.511 263.7740454710893',
 '0.0175 261.833 262.1069015172529',
 '0.02 259.908 260.1952231766013',
 '0.0225 257.743 258.04403838638797']
```

1 行目が header で、1 列目が q、2 列目が $I(q)$、3 列めが $I_{fit}(q)$ という構造になっている。

ATSAS では `*.fit` という拡張子がついているファイル形式であり、普通にそのまま plot すると 3 列めが見えず、ファイルをプロットしても全く同じ ?? などということが起こる。

$P(r)$ に関しては

[2]:
```
!!head 6lyz_pr_calc_1.dat
```

[2]:
```
['#',
 '0.0 0.0 0.0',
 '0.4464 2.766541091949083 0.0',
 '0.8929 5.553805338144977 0.0',
 '1.339 8.37560817475222 0.0',
 '1.786 11.247491967455916 0.0',
 '2.232 14.195360659064583 0.0',
 '2.679 17.245118192386734 0.0',
 '3.125 20.429576228313145 0.0',
 '3.571 23.779819498214028 0.0']
```

さきほど同様、`*.fit` というファイル形式で、

1 行目が header で、1 列目が r、2 列目が $P(r)$、3 列めが $P_{fit}(r)$ という構造になっている。

散乱強度にせよ、動径分布関数にせよ **“*.fit“** のファイル形式では **2** 列めと **3** 列めの差分をプロットするのが普通である。

散乱曲線の比較

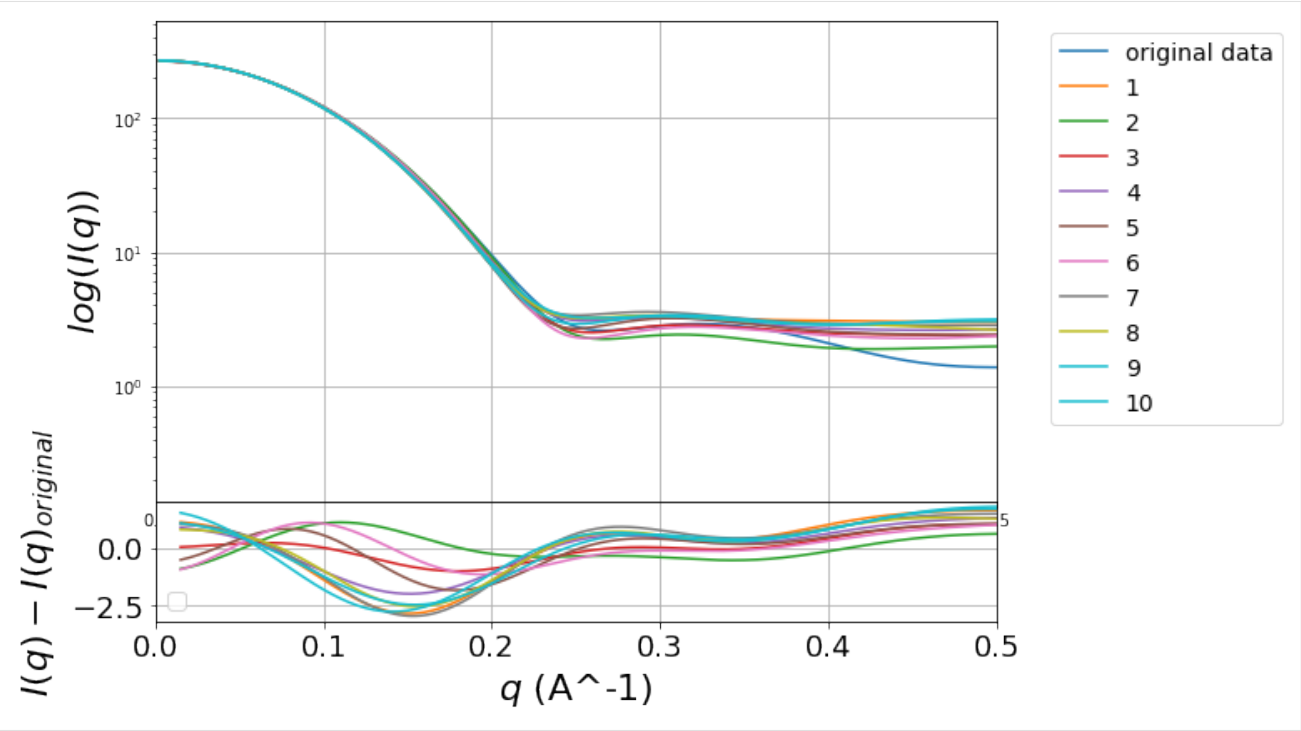
以下のセルを実行すればモデルから予想されるフィットデータと入力ファイルの比較ができる。ここで、

グラフの上部：Original data と Output data の散乱曲線がプロットされる。

グラフの下部：Original data と Output data の散乱曲線の log(I(q)) の差分がプロットされる。

[2]:
```
# -*- coding: utf-8 -*-
"""
Created on Thu Dec 13 14:17:03 2018

@author: Takeuchi
modified by Fujisawa
"""

import tkinter
from tkinter import messagebox as tkMessageBox
from tkinter import filedialog as tkFileDialog
from tkinter import simpledialog as tkSimpleDialog
import os,pylab
import matplotlib.pyplot as plt

root=tkinter.Tk()
root.withdraw()

dir0=os.getcwd()
infiles=tkFileDialog.askopenfilenames(title='q-log(I(q)) plot',initialdir='.',filetypes=[("dat file
 ↪","*.dat")])
dir1=os.path.dirname(infiles[0])
os.chdir(dir1)

fig = pylab.figure(figsize=(7,5))
#fig = pylab.figure()
# サブプロットを 8:2 で分割
ax1 = fig.add_axes((0, 0.2, 1, 0.8))
ax2 = fig.add_axes((0, 0, 1, 0.2), sharex=ax1)

# 散布図の x 軸のラベルとヒストグラムの y 軸のラベルを非表示
#ax1.tick_params(labelbottom="off")
#ax2.tick_params(labelleft="off")

ax1.set_ylabel(r'$log(I(q))$',fontsize=22)
ax2.set_ylabel(r'$I(q)-I(q)_{original}$',fontsize=22)
ax2.set_xlabel(r"$q$ (A^-1)",fontsize=22)
#ax1.tick_params(labelsize=18)
ax2.tick_params(labelsize=18)

cmap = plt.get_cmap("tab10") # ココがポイント

for i,infn in enumerate(infiles):
    pdata=pylab.loadtxt(infn,skiprows=1)
    st='label for %s' %os.path.basename(infn)
    plabel=st
    plabel=tkSimpleDialog.askstring('label for plot',os.path.basename(infn))
    if i==0:
        ax1.semilogy(pdata[5:,0],pdata[5:,1],color=cmap(0),label='original data')
        ax1.semilogy(pdata[:,0],pdata[:,-1],color=cmap(1+i),label=plabel)
    else:
        ax1.semilogy(pdata[:,0],pdata[:,-1],color=cmap(1+i),label=plabel)
    psub=pdata[5:,-1]-pdata[5:,1]
    ax2.plot(pdata[5:,0],psub,color=cmap(1+i))

ax1_ymax=max(pdata[:,1])*2
ax1_ymin=pdata[-1,1]*0.1
ax2.set_xlim(0.0,pdata[-1,0])
ax1.set_ylim(ax1_ymin,ax1_ymax)
ax1.grid()
ax2.grid()
ax1.legend(bbox_to_anchor=(1.05, 1), loc='upper left',fontsize=14)
ax2.legend(fontsize=14)
pylab.show()

os.chdir(dir0)
```

```
No handles with labels found to put in legend.
```



動径分布関数の比較

以下のセルを実行すればモデルから予想されるフィットデータと入力ファイルの比較ができる。ここで、

グラフの上部：Original data と Output data の動径分布関数がプロットされる。

グラフの下部：Original data と Output data の動径分布関数の $P(r)$ の差がプロットされる。

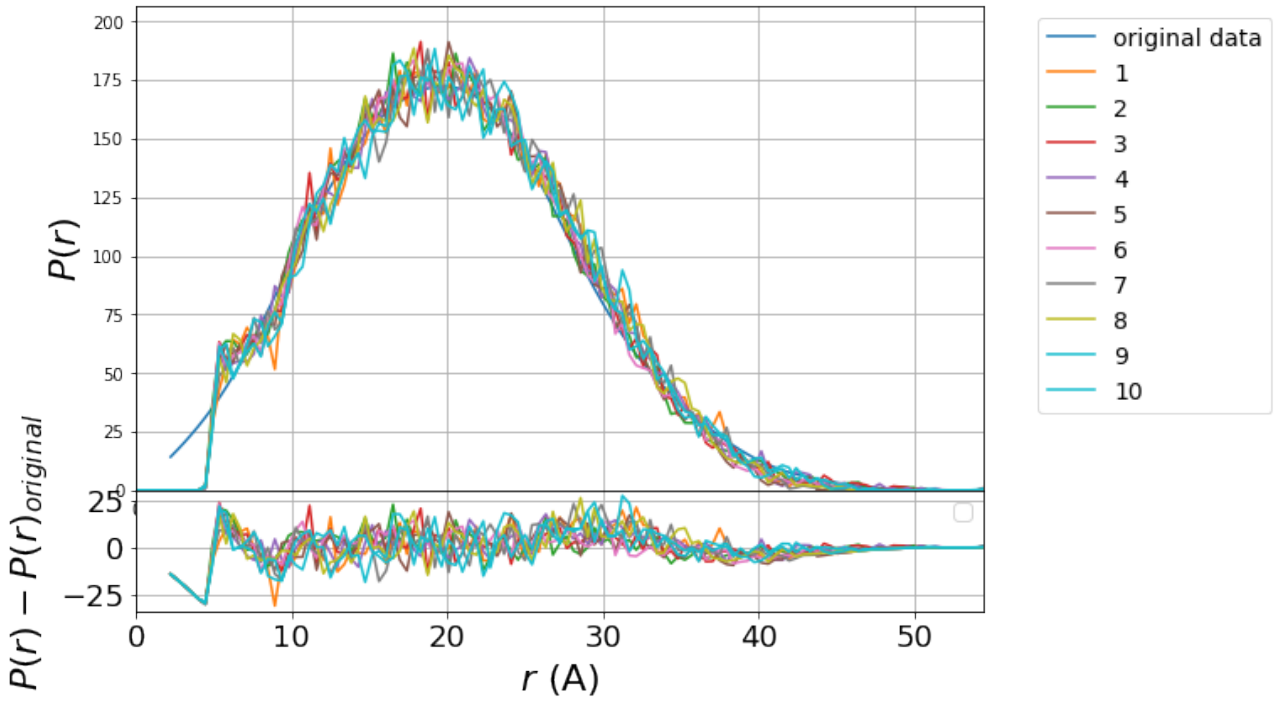
[3]:
```
# -*- coding: utf-8 -*-
"""
Created on Thu Dec 13 14:17:03 2018

@author: Takeuchi
modified by Fujisawa
"""

import tkinter
from tkinter import messagebox as tkMessageBox
from tkinter import filedialog as tkFileDialog
from tkinter import simpledialog as tkSimpleDialog
import os,pylab
import matplotlib.pyplot as plt

root=tkinter.Tk()
root.withdraw()

dir0=os.getcwd()
infiles=tkFileDialog.askopenfilenames(title='q-log(I(q)) plot',initialdir='.',filetypes=[("dat file
↪","*.dat")])
dir1=os.path.dirname(infiles[0])
os.chdir(dir1)

fig = pylab.figure(figsize=(7,5))
#fig = pylab.figure()
# サブプロットを 8:2 で分割
ax1 = fig.add_axes((0, 0.2, 1, 0.8))
ax2 = fig.add_axes((0, 0, 1, 0.2), sharex=ax1)

# 散布図の x 軸のラベルとヒストグラムの y 軸のラベルを非表示
#ax1.tick_params(labelbottom="off")
#ax2.tick_params(labelleft="off")

ax1.set_ylabel(r'$P(r)$',fontsize=22)
ax2.set_ylabel(r'$P(r)-P(r)_{original}$',fontsize=22)
ax2.set_xlabel(r"$r$ (A)",fontsize=22)
#ax1.tick_params(labelsize=18)
```

(次のページに続く)

(前のページからの続き)

```
ax2.tick_params(labelsize=18)

cmap = plt.get_cmap("tab10") # ココがポイント

for i,infn in enumerate(infiles):
    pdata=pylab.loadtxt(infn,skiprows=1)
    st='label for %s' %os.path.basename(infn)
    plabel=st
    plabel=tkSimpleDialog.askstring('label for plot',os.path.basename(infn))
    if i==0:
        ax1.plot(pdata[5:,0],pdata[5:,1],color=cmap(0),label='original data')
        ax1.plot(pdata[:,0],pdata[:,-1],color=cmap(1+i),label=plabel)
    else:
        ax1.plot(pdata[:,0],pdata[:,-1],color=cmap(1+i),label=plabel)
    psub=pdata[5:,-1]-pdata[5:,1]
    ax2.plot(pdata[5:,0],psub,color=cmap(1+i))

ax1_ymax=max(pdata[:,1])*1.2
ax1_ymin=0.0
ax2.set_xlim(0.0,pdata[-1,0])
ax1.set_ylim(ax1_ymin,ax1_ymax)
ax1.grid()
ax2.grid()
ax1.legend(bbox_to_anchor=(1.05, 1), loc='upper left',fontsize=14)
ax2.legend(fontsize=14)
pylab.show()

os.chdir(dir0)
```

```
No handles with labels found to put in legend.
```



2.4 ビーズモデルの統計処理

タンパク質溶液散乱のビーズモデル計算はそれが $I(q)$ に対してであれ、$P(r)$ に対してであれ、確率論的なアルゴリズムを用いるので若干の形の変動がある。そこで、ビーズモデルの検証として以下の操作を行う。 [6]

1. 試行回数分生成されたビーズモデルを平均化し、代表モデルの算出
2. 代表モデルから計算された散乱曲線、動径分布関数の検証
3. 結晶構造との比較

1. 代表ビーズモデルの計算

ビーズモデル 10 個から DAMAVER を用いて平均像作成

SHAPES を実行するとビーズモデルの出力は素のビーズモデルと分子体積で整形したビーズモデルの 2 種類が生成される。ここでは原論文とは違い ビーズモデル *_beads_#.pdb を元に代表ビーズモデルを計算する。

この計算には DAMAVER という ATSAS のプログラムがあるのでそれを使用する。

[1]:
```
import tkinter
from tkinter import messagebox as tkMessageBox
from tkinter import filedialog as tkFileDialog
from tkinter import simpledialog as tkSimpleDialog
import os
```

[ ]:
```
dir0=os.getcwd()
root = tkinter.Tk()
root.withdraw()
infiles=tkFileDialog.askopenfilenames(title='Beads PDB files from shape',filetypes=[("ビーズファイル",
 ↪"*beads*.pdb")])
dir1=os.path.dirname(infiles[0])
os.chdir(dir1)
fns=""
for f in infiles:
    fns=fns+' '+os.path.basename(f)
cmd='damaver -a %s' %fns
print (cmd)
!damaver -a $fns
```

[ ]:
```
damaver -a  6lyz_beads_1.pdb 6lyz_beads_2.pdb 6lyz_beads_3.pdb 6lyz_beads_4.pdb 6lyz_beads_5.pdb␣
 ↪6lyz_beads_5r.pdb 6lyz_beads_6.pdb 6lyz_beads_7.pdb 6lyz_beads_8.pdb 6lyz_beads_9.pdb 6lyz_beads_
 ↪10.pdb
                :
            途中 略
                :

 Wrote file ..................................... : 6lyz_beads_10r.pdb
 Read file ...................................... : 6lyz_beads_5.pdb
 Read file ...................................... : 6lyz_beads_5rr.pdb
 Read file ...................................... : 6lyz_beads_7r.pdb
 Read file ...................................... : 6lyz_beads_8r.pdb
 Read file ...................................... : 6lyz_beads_6r.pdb
 Read file ...................................... : 6lyz_beads_2r.pdb
 Read file ...................................... : 6lyz_beads_3r.pdb
 Read file ...................................... : 6lyz_beads_4r.pdb
 Read file ...................................... : 6lyz_beads_9r.pdb
 Read file ...................................... : 6lyz_beads_1r.pdb
 Read file ...................................... : 6lyz_beads_10r.pdb
 Wrote file ..................................... : damaver.pdb
 Number of atoms written ........................ : 1702
 Read file ...................................... : damaver.pdb
 Number of atoms ................................ : 1702
 Number of phases ............................... : 1
 Minimum number of contacts ..................... : 3
 Maximum number of contacts ..................... : 13
 Selected contact threshold ..................... : 5
 Atomic radius .................................. : 1.750
 Excluded volume per atom ....................... : 30.34
 Maximum radius ................................. : 29.80
 Average excluded volume ........................ : 0.0
 Selected cut-off volume ........................ : 2.5817E+04
 Final contact threshold ........................ : 5
 Final cut-off volume ........................... : 2.5817E+04
 Final number of atoms .......................... : 853
 Final volume ................................... : 2.5877E+04
 Wrote file ..................................... : damfilt.pdb
 Read file ...................................... : damaver.pdb
 Number of atoms ................................ : 1702
 Number of phases ............................... : 1
 Minimum number of contacts ..................... : 3
 Maximum number of contacts ..................... : 13
 Selected contact threshold ..................... : 5
 Atomic radius .................................. : 1.750
 Excluded volume per atom ....................... : 30.34
 Maximum radius ................................. : 29.80
 Average excluded volume ........................ : 0.0
 Selected cut-off volume ........................ : 2.5817E+04
 Final contact threshold ........................ : 5
 Final cut-off volume ........................... : 2.5817E+04
 Final number of atoms .......................... : 853
 Final volume ................................... : 2.5877E+04
 Wrote file ..................................... : damstart.pdb
```

damaver を実行すると

- damaver.pdb
- damstart.pdb
- damfilt.pdb

などのビーズモデルファイルが出力されるが、代表的なビーズモデルは damfilt.pdb である。

代表モデルから計算された散乱曲線、動径分布関数の検証

damfilt.pdb から $I(q)$ や $P(r)$ を計算

ビーズモデルでは各試行計算の結果では非常によくモデルの $I(q)$ や $P(r)$ と入力値が一致するが、各試行回の結果を平均化して代表モデルを計算するとずれることが多い。

実は、元論文では記載されていないが、分子体積で整形されたビーズモデルで代表モデルを算出し、その damfilt.pdb から $I(q)$ や $P(r)$ を計算するとかなりずれてしまう。

筆者は念の為、**いつも代表モデルから計算された散乱曲線、動径分布関数の検証をするようにしている。**

damfilt.pdb から $I(q)$ や $P(r)$ を計算するために、以下のコード damfilt_analysis を用意した。

[1]:
```
def damfilt_analysis(pdbfile_in,aList_r,aList_pr,aList_pr_model,inFile,prefix,aList_q,aList_i,aList_
↪i_calc):

# written by Hinami Suzuki
# This code reguires shapes_module.py
#

    import shapes_module as sm


#damfilt.pdb のそれぞれのビーズの位置を取得
    aList_beads_x=[]
    aList_beads_y=[]
    aList_beads_z=[]

    sm.read_pdb(aList_beads_x,aList_beads_y,aList_beads_z,pdbfile_in)

#GNOM 出力データからオリジナルの r,P(r),angstrom_scale,num_hist を取得
    #aList_r = []
    #aList_pr = []
    aList_pr_sd = []
    #aList_pr_model = []
    aList_pr_model_test = []
    aList_pr_model_test2 = []

    dmax,hist_grid,num_hist,angstrom_scale=sm.read_pr(aList_r,aList_pr,aList_pr_sd,aList_pr_model,
↪aList_pr_model_test,aList_pr_model_test2,inFile)

#damfilt.pdb のビーズの位置とオリジナルの hist_grid から damfilt.pdb の P(r) 取得
    sm.calc_pr(aList_beads_x,aList_beads_y,aList_beads_z,aList_pr_model,hist_grid)

#オリジナルと damfilt.pdb の P(r) のスケールを合わせる
    sm.scale_pr(aList_pr,aList_pr_sd,aList_pr_model)

#r, オリジナルの P(r),damfilt.pdb から計算された P(r) を dat ファイルに書き込む
    outfile_pr = prefix + 'damfilt_pr_calc_' + '.dat'

    sm.pr_writer(aList_pr,aList_r,aList_pr_model,outfile_pr)

    aString = 'Output p(r) file: ' + str(outfile_pr)
    print (aString)

#GNOM 出力ファイルからオリジナルの q,I(q) を取得
    aList_i_sd=[]
    aList_i_reg=[]
    sm.read_i(aList_q,aList_i,aList_i_sd,aList_i_reg,inFile,angstrom_scale)
```

(次のページに続く)

(前のページからの続き)

```
#damfilt.pdb の計算された P(r) から damfilt.pdb の I(q) を計算
    nbeads=len(aList_beads_x)
    sm.ft_to_intensity(aList_q,aList_i_calc,aList_r,aList_pr_model,nbeads)

    (chi_sq,rvalue) = sm.score_Ic(aList_q,aList_i,aList_i_sd,aList_i_calc)

#オリジナルと damfilt.pdb の I(q) のスケールを合わせる
    aString = 'Rvalue: ' + str('%4.3f'%(rvalue)) + ' CHI-squared: ' + str('%4.3f'%(chi_sq))
    print (aString)

#q, オリジナルの I(q),damfilt.pdb から計算された I(q) を dat ファイルに書き込む
    file_intensity = prefix + 'damfilt_i_calc_' + '.dat'
    sm.write_all_data(file_intensity,aList_q,aList_i,aList_i_calc,aString)

    aString = 'Output intensity file: ' + str(file_intensity)
    print (aString)

    return;
```

以下で実際に上の関数を用いて計算を行う。

- 入力：ビーズモデルの代表 (PDB ファイル)、元の GNOM 出力ファイル
- 出力：damfilt_pr_calc_.dat、damfilt_i_calc_.dat

[2]:
```
%matplotlib inline
import tkinter,os
import pandas as pd
import matplotlib.pyplot as plt
from tkinter import messagebox as tkMessageBox
from tkinter import filedialog as tkFileDialog
from tkinter import simpledialog as tkSimpleDialog

root=tkinter.Tk()
root.withdraw()

#入出力の配列などは準備しておく
prefix=''
aList_r = []
aList_pr = []
aList_pr_model=[]
aList_q=[]
aList_i=[]
aList_i_calc=[]
#
#ファイルの読み込み
pdbfile_in=tkFileDialog.askopenfilename(title='Open damfilt file',filetypes=[("pdb ファイル",
 ↪"*damfilt.pdb")])
inFile=tkFileDialog.askopenfilename(title='Open gnom file',filetypes=[("アウトファイル","*.out")])
print('Input pdb file:',os.path.basename(pdbfile_in))
print('Original gnom file:',os.path.basename(inFile))
damfilt_analysis(pdbfile_in,aList_r,aList_pr,aList_pr_model,inFile,prefix,aList_q,aList_i,aList_i_
 ↪calc)
```

```
Input pdb file: damfilt.pdb
Original gnom file: 6lyz.out
Output p(r) file: damfilt_pr_calc_.dat
Rvalue: 0.018 CHI-squared: 33.905
Output intensity file: damfilt_i_calc_.dat
```

ファイルから直接プロットしてもよいのだが Pandas の Dataframe の形に変換しておく。

[3]:
```
df = pd.DataFrame({'r': aList_r,'P(r)_original': aList_pr,'P(r)_beadsmodel':aList_pr_model})
df.head()
```

[3]:

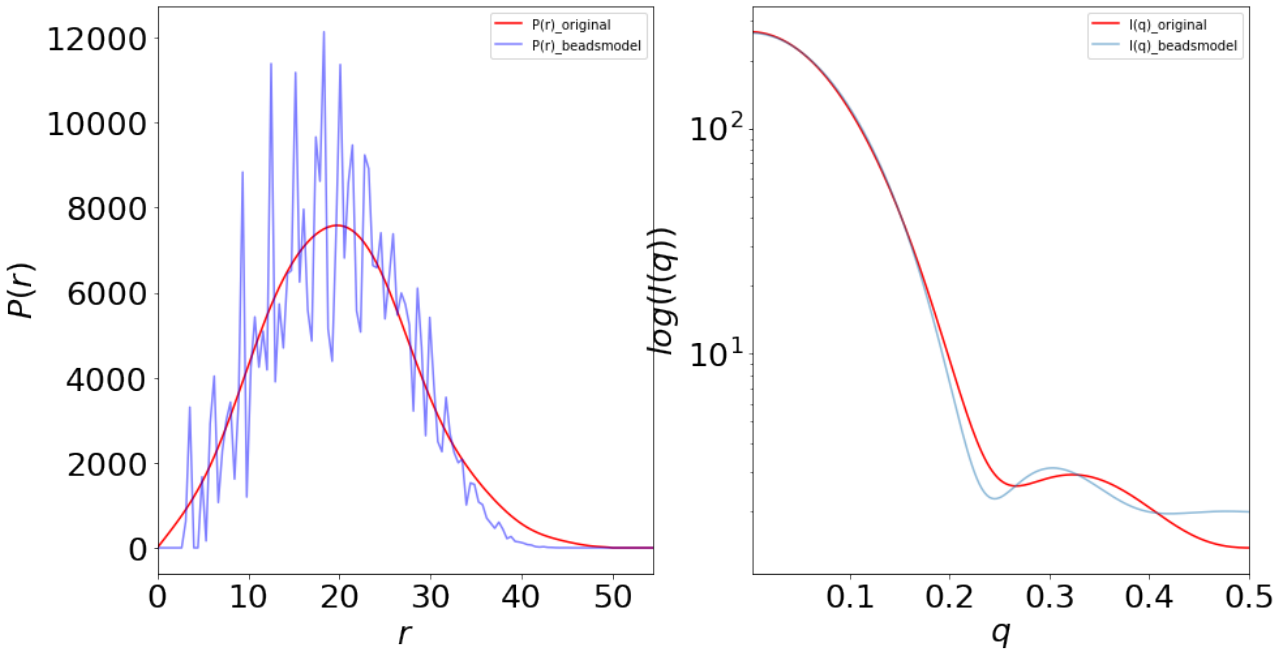
	r	P(r)_original	P(r)_beadsmodel
0	0.0000	0.000000	0.0
1	0.4464	121.766009	0.0
2	0.8929	244.444123	0.0
3	1.3390	368.642411	0.0
4	1.7860	495.044954	0.0

[4]:
```
df2 = pd.DataFrame({'q': aList_q, 'I(q)_original': aList_i,'I(q)_beadsmodel':aList_i_calc})
df2.head()
```

[4]:

	q	I(q)_original	I(q)_beadsmodel
0	0.0025	268.083	265.125140
1	0.0050	267.688	264.760052
2	0.0075	267.032	264.152562
3	0.0100	266.115	263.304156
4	0.0125	264.940	262.216902

[5]:
```
fig = plt.figure(figsize=(15, 8))

ax1 = fig.add_subplot(121)
df.plot(ax=ax1,x='r',y='P(r)_original',color='red',label='P(r)_original',fontsize=25)
df.plot(ax=ax1,x='r',y='P(r)_beadsmodel',color='blue',alpha=0.5,label='P(r)_beadsmodel',fontsize=25)
ax1.set_xlabel('$r$',fontsize=25)
ax1.set_ylabel('$P(r)$',fontsize=25)

ax2 = fig.add_subplot(122)
df2.plot(ax=ax2,x='q',y='I(q)_original',color='red',label='I(q)_original',fontsize=25,logy=True)
df2.plot(ax=ax2,x='q',y='I(q)_beadsmodel',alpha=0.5,label='I(q)_beadsmodel',fontsize=25,logy=True)
ax2.set_xlabel('$q$',fontsize=25)
ax2.set_ylabel('$log(I(q))$',fontsize=25)

#fig.savefig('damfilt.png')
```



結晶構造との比較

モデルのばらつきによるモデルの分解能の評価

結晶構造の比較する前に、モデルの分解能について調べてみる。

モデルの分解能は Fourier Shell Correlation (FSC) でビーズモデルのばらつきから、モデルの分解能がわかるという報告がある。[7]

これは、`damaver` を実行した際に生成されるログファイル `damsel.log` に記述されている。

[1]:
```
flog=open('damsel.log','r')
lines=flog.readlines()
res = [i for i in lines if 'Ensemble Resolution' in i]
print(res[0])
```

```
    Ensemble Resolution =     32 +-   3 Angstrom
```

元の pdb ファイルとの重ね合わせ

結晶構造と比較するには代表的ビーズモデルの向きをあわせてやる（アラインメントしてあげる）必要がある。

`6lyz.pdb` に `damfilt.pdb` をアラインメントしてみよう。

ここでは supalm [8] という ‘‘ATSAS‘‘のプログラムを使用する。

[2]:
```
!!supalm  6lyz.pdb damfilt.pdb
```

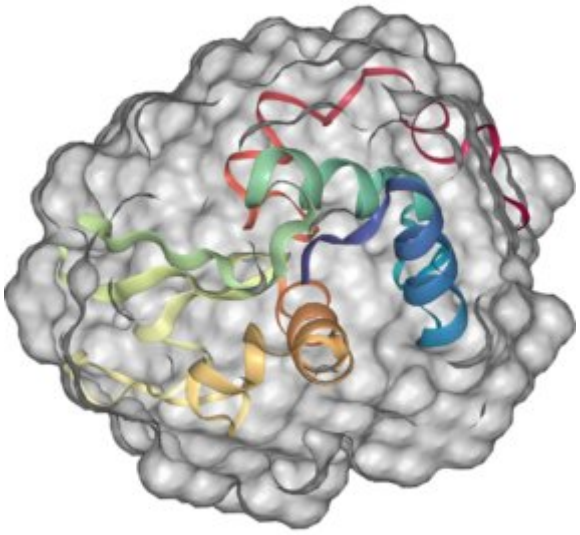
[2]:
```
[' Read file ........................................... : 6lyz.pdb',
 ' Number of atoms read ................................ : 1102',
 ' Fineness of the template ............................ : 1.514',
 ' Read file ........................................... : damfilt.pdb',
 ' Number of atoms read ................................ : 853',
 ' Fineness of the superimposed structure .............. : 3.500',
 '',
 '  Principal axes orientation table',
 '',
 '    Distance                       Orientation',
 '',
 '   1.1360454524805610                1          -1          -1',
 '   1.1419180336932468                1           1          -1',
 '   1.1706840733884827               -1           1           1',
 '   1.1720072826064973               -1          -1           1',
 '   1.1741133266525163                1           1           1',
 '   1.1992066476404875               -1          -1          -1',
 ' -----------------------------------------',
 '   1.1996296950877527                1          -1           1',
 '   1.2065759032678545               -1           1          -1',
 ' Calculating amplitudes from 6lyz.pdb',
 ' --WARNING: Recompute the intensity with LM= 13',
 ' Calculating amplitudes from damfilt.pdb',
 '                              Initial       Final',
 '              Orientation   Correlation Correlation',
 '              1    -1    -1     0.87072     0.87756',
 '              1     1    -1     0.87217     0.87629',
 '             -1     1     1     0.86609     0.87156',
 '             -1    -1     1     0.86081     0.87057',
 '              1     1     1     0.86799     0.88473',
 '             -1    -1    -1     0.85997     0.86985',
 ' Final distance (NSD) =    1.1673967678946129     ',
 ' Wrote file ......... :    damfiltr.pdb',
 '']
```

6lyz.pdb にアラインメントされた damfilt.pdb は damfiltr.pdb として保存されているのでそれを読み込む。

通常なら別のプログラムを立ち上げて確認することになるのだが、jupyter-notebook 上で 3 D 表示し、視点を変えることもできる。

[3]:
```
import nglview
view = nglview.show_file("damfiltr.pdb")   # ビーズファイルを指定する
view.add_surface('.CA', opacity=0.1)   # ビーズファイルの表面だけ表示する
#以下の部分はグラフィックボードとの相性で同時に表示されないこともあるのでコメントアウトしている。
mol2=nglview.FileStructure("6lyz.pdb") #pdb ファイルを指定する
view.add_structure(mol2)
view.camera = 'orthographic'
view.center()
view
```

```
NGLWidget()
```



気に入った向きで画像を動かして、次のコマンドで保存する。

```
[4]: view.download_image('beads.png')
```

Normalized Spatial Discrepancy (NSD) 値

構造の差異を表す指標として RMSD（Root Mean Square Deviaiton) 平均自乗偏差が良く使われるが、結晶構造とビーズモデルでは分解能が違うので RMSD は使えない。

そこで、タンパク質溶液散乱では NSD(Normalized Spatial Discrepancy) という指標を使用する。[6]

NSD は、元の PDB ファイルとの重ね合わせでも NSD が最小になるよう PDB の方向を変えているので、`damfiltr.pdb` ファイルにも記載されている。

```
[5]: flog=open('damfiltr.pdb','r')
     lines=flog.readlines()
     res = [i for i in lines if 'Final distance' in i]
     print(res[0])
```

```
REMARK 265  Final distance (NSD)      :    1.1674
```

以上、データ解析の実際を見てきた。 **jupyter-notebook** の活用の注意点を私なりにあげてみると、

重要: jupyter-notbook の注意点

- **jupyter-notebook** はプログラムを開発するのには不向きで、基本的に確立したプログラムを実行する環境と捉えるべき。
- notebook で使われる変数は全てのセルに共通のグローバル変数なので、python コードを呼び出す前に戻り値の変数を準備しておく。
- notebook を再実行する際はそのたび毎に kernel の reset をして再実行したほうが良い。

- CUI(Command User Interface) での対話的なコードは実行できない。入力値はセルに書き込んで記録に残すようにすると良い。ただし、ファイル名の入出力に関しては本稿のように GUI を使う方が楽である。

2.5 より発展的な話題

.

対話型プログラムの実行

jupyter-notebook では CUI(Command User Interface) での対話的なプログラムの実行には向いていない。**ATSAS** のいくつかのプログラム、例えば先程あげた **crysol** などにはコマンドの引数として与えることができないパラメータなどが対話モードで設定できる。

表 3: **crysol** のコマンドラインにおけるオプション [3]

オプション	オプションの説明	default 値
`-lm`	球面調和関数の最大次数。散乱体の分解能を定義する。	15
`-fb`	散乱体の表面を記述するフィボナッチ数を定義するもの。	17
`-sm`	散乱ベクトルの最大値	0.5
`-dns`	溶媒の電子密度	0.334(e/A^3
`-dro`	水和層のコントラスト	0.03(e/A^3)
`-cst`	buffer 差分補正。定数差分による補正を行う。	-0.5 から+0.5

以下に主要なパラメータの説明を行う。

Dro :水和層のコントラスト。前述の $\Delta\rho_b$ に相当する。デフォルト値は 0.03 (e/A^3)

Ra :原子団の半径。この場合は水和層の厚みに相当する。デフォルト値は 1.62(A)

ExVol :排除体積。crysol によって計算されたタンパク質の体積に相当する。デフォルト値は BSA の場合は 82810 (A^3)

dns :溶媒の電子密度 $\rho_{solvent}$ 。デフォルト値は 0.334 (e/A^3)

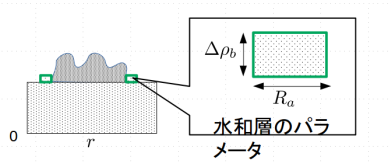


図 3: **crysol** の計算パラメータの説明

実験データと結晶構造との理論値を比較する際、一般にデフォルトの **crysol** のパラメータでは、溶媒等が水でないので直接比較できない。そこで、実験データを与えると、理論曲線のパラメータ (特に `Dro` , `Ra` , `ExVol`) を微調してできるだけ実際の実験データをうまく説明するような理論曲線を計算し、それでもうまくフィットできないときには実際の構造が違うと判断する。

高圧セルなどの特殊なセルを使用する時は、これらの計算パラメータを細かく制御する必要がでてくる。

crysol の対話モードでの実行例

ここでは、 水和水の密度コントラスト Dro だけ対話モードで微調設定することを考える。

実際に **crysol** を実行させた例を見てみよう。例えば、リゾチウムというタンパク質の溶液散乱の実験値　lyzexp.dat と結晶構造の A 鎖 6lyz.pdb との比較を行うとする。

1. crysol の実行すると以下の様に表示される。

```
***  ------------------------------------------------  ***
***    C R Y S O L  Wintel/UNIX/Linux version 2.8.3    ***
***  Please reference: D.Svergun, C.Barberato          ***
***   & M.H.J.Koch (1995) J. Appl.Cryst., 28, 768-773  ***
***   Version (LMAX=50) for small and wide angles      ***
***        Last revised  ---  12/08/15 14:00           ***
***   Copyright (c) ATSAS Team                         ***
***   EMBL, Hamburg Outstation, 1995 - 2015            ***
***  ------------------------------------------------  ***
```

2. 各オプションと pdb ファイルの入力

   各オプションと PDB ファイルを入力する。

```
------------------------------------------------
            Program options :
0 - evaluate scattering amplitudes and envelope
1 - evaluate only envelope and Flms
2 - read CRYSOL information from a .sav file
Type crysol -help for batch mode use
------------------------------------------------

Enter your option ..................... <            0 >:
Brookhaven file name .................. <         .pdb >: 6lyz
------------------------------------------------

------------------------------------------------
        Following file names will be used:
6lyz00.log -- CRYSOL log-file          (ASCII)
6lyz00.sav -- save CRYSOL information  (binary)
6lyz00.flm -- multipole coefficients   (ASCII)
6lyz00.int -- scattering intensities   (ASCII)
6lyz00.fit -- fit to experimental data (ASCII)
6lyz00.alm -- net partial amplitudes   (binary)
------------------------------------------------

------------------------------------------------
Maximum order of  harmonics ........... <           15 >:
Order of Fibonacci grid ............... <           17 >:
----------  Reciprocal space grid  -------------
( in s = 4*pi*sin(theta)/lambda [1/angstrom] )
Maximum s value ....................... <        .5000 >:
Number of points ...................... <           51 >: 201
Account for explicit hydrogens? [ Y / N ] <          No >:
Read atoms and evaluate geometrical center ...
Number of atoms read ................................ : 1001
Number of discarded waters .......................... : 101
Percent processed      10  20  30  40  50  60  70  80  90 100
Processing atoms   :>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
Center of the excess electron density:  0.452 -0.004  0.276
Processing envelope:>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
Processing envelope:>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
Coefficients   saved to file 6lyz00.flm
CRYSOL data    saved to file 6lyz00.sav
```

3. 計算された各値が表示される。

```
--- Structural parameters (sizes in angstroms) ---
Electron   rg   :  13.99       Envelope   Rg      :  14.08
Shape      Rg   :  13.97       Envelope  volume   : 0.1806E+05
Shell    volume : 0.1129E+05   Envelope  surface  :  3150.
Shell      Rg   :  18.98       Envelope  radius   :  25.49
Shell    width  :  3.000       Envelope  diameter :  49.04
Molecular weight: 0.1432E+05   Dry volume         : 0.1736E+05
Displaced volume: 0.1741E+05   Average atomic rad.:  1.607
Number of residuals :   129
```

1-3 は常に共通である。

4. 実験データをフィットさせる選択を行う。また、実験データファイルを入力する。

```
Fit the experimental curve [ Y / N ] .. <          Yes >:
Enter data file ....................... <         .dat >: lyzexp.dat
```

5. buffer 差分補正を実行するかの選択。今回は実行しないので N とする。

```
Subtract constant ..................... <           no >:
Maximum angle in the data file ........................ : 0.4984
Number of experimental points ......................... : 197
Angular units in the input file:
4*pi*sin(theta)/lambda [1/angstrom] (1)
4*pi*sin(theta)/lambda [1/nm]       (2)
2 *  sin(theta)/lambda [1/angstrom] (3)
2 *  sin(theta)/lambda [1/nm]   (4) ..... <            1 >:
Angular units multiplied by ........................... : 1.000
```

6. 溶媒の電子密度値の入力

```
Electron density of the solvent, e/A**3  <       0.3340 >:
Number of experimental points used .................... : 197
```

7. 計算パラメータの仮最適化された値が出力される。

   ここでは、 Dro , Ra , ExVol の 3 つのパラメータが同時に最適化されている。

```
------  Fitting the experimental data ...    ---
6lyz.pdb  Dro:0.010  Ra:1.460  Vol: 17762.  Chi^2: 0.202
Plot the fit [ Y / N ] ................ <          Yes >: N
```

8. 更なる設定を行うかの確認。今回は行うので Y とする。

```
Another set of parameters [ Y / N ] .... <           No >: Y
Minimize again with new limits [ Y / N ] <           No >: Y
```

9. 計算パラメータの入力

```
Average atomic radius is ............................... : 1.607
Minimum radius of atomic group ......... <        1.400 >: 1.607
Maximum radius of atomic group ......... <        1.800 >: 1.607
Smax in the fitting range .............. <       0.4984 >:
Solvent density is ..................................... : 1.000E-02
Minimum contrast in the shell .......... <        0.000 >:
Maximum contrast in the shell .......... <  7.5000E-002 >:
Van der Waals volume is ................................ : 1.7762E+04
Minimum excluded volume ................ <  1.6108E+004 >: 1.7762E4
Maximum excluded volume ................ <  1.8720E+004 >: 1.7762E4
Subtract constant ...................... <           no >:
Number of experimental points used ..................... : 197
```

   探索の上限下限に同一の計算パラメータを指定すれば、計算パラメータの最適化はされず、各計算パラメータは指定値で計算され、実験データに対してスケーリングのみした散乱曲線が得られる。

   ここでは、

   - Average atomic radius is として表示された 1.607 の値を読み込み、Minimum , Maximum にその値を代入する。
   - Van der Waaks volume is として表示された 1.7762E+04 の値を読み込み、Minimum , Maximum にその値を代入する。

   の 2 点の処理を行えば、 Dro だけ最適化されて実験データにフィットするような動作となる。

   デフォルトの Ra , ExVol の計算パラメータ値で Dro だけ最適化されて理論散乱曲線を生成し、実験データとのスケールだけを行う。

10. スケーリング結果の出力

```
------  Fitting the experimental data ...    ---
6lyz.pdb  Dro:0.013  Ra:1.607  Vol: 17762.  Chi^2: 0.203
Plot the fit [ Y / N ] ................. <          Yes >: N
Another set of parameters [ Y / N ] .... <           No >:
Rg from the slope of net intensity ..................... : 14.89
Average electron density ............................... : 0.4298
Data fit       saved to file 6lyz00.fit
Intensities    saved to file 6lyz00.int
I_abs(s)[cm^-1]/c[mg/ml] saved to file 6lyz00.abs
Net amplitudes saved to file 6lyz00.alm
```

これらの入力しないといけないステップをまとめると

表 4: **crysol** の対話モードにおける入力リスト

2-1.	Enter your option ..... < 0 >:	
2-2.	Brookhaven file name ..... < .pdb >:	6lyz
2-3.	Maximum order of harmonics ..... < 15 >:	
2-4.	Order of Fibonacci grid ..... < 17 >:	
2-5.	Maximum s value ..... < .5000 >:	
2-6.	Number of points ..... < 51 >:	201
2-7.	Account for explicit hydrogens? [Y / N] < No >:	
4-1.	Fit the experimental curve [Y / N] .. < Yes >:	
4-2.	Enter data file ..... < .dat >:	lyzexp.dat
5-1.	Subtract constant ..... < no >:	N
5-2.	2 * sin(theta)/lambda [1/nm] (4) ..... < 1 >:	
6	Electron density of the solvent e/A**3 < .3340 >:	
7	Plot the fit [Y / N] ..... < Yes >:	N
8-1.	Another set of parameters [Y / N] .... < No >:	Y
8-2.	Minimize again with new limits [Y / N] < No >:	Y
9-1.	Minimum radius of atomic group ..... < 1.400 >:	1.607
9-2.	Maximum radius of atomic group ..... < 1.800 >:	1.607
9-3.	Smax in the fitting range ..... < 0.4984 >:	
9-4.	Minimum contrast in the shell ..... < .000 >:	
9-5.	Maximum contrast in the shell ..... < 7.5000E-002 >:	
9-6.	Minimum excluded volume ..... < 1.6108E+004 >:	17800
9-7.	Maximum excluded volume ..... < 1.8720E+004 >:	17800
9-8.	Subtract constant ..... < no >:	
10-1.	Plot the fit [Y / N] ..... < Yes >:	N
10-2.	Another set of parameters [Y / N] .... < No >:	

例えば、 `ExVol` の値を 9-5. の後に表示された値、 `1.7762E+04` を代入しようとすると、

プログラムの出力を読み取って、それに基づいて別の入力を行う

ことが要求される。これは単純に入力テキストを入れたファイルを準備して実行するというわけにはいかない。

このようなプログラムの自動応答の python モジュールに `pexpect` がある。

https://pexpect.readthedocs.io/en/stable/

pexpect の使い方

- **crysol** の起動の仕方

```
p = pexpect.spawn("crysol")
```

- **crysol** への入力の仕方

ATSAS は入力を促す際に必ず

`>:`

で質問文が終わるので、

`p.expect(r">:")`

でプログラムの出力で `>:` の部分を待つ。それが来たら

`p.send($str)`

で文字列 `$str` をプログラムに入力すれば良い。

- **crysol** からの出力の受け方

  例えば数字の出力の次の行の先頭が `Minimum` で始まるとしたら

  `p.expect(r"Minimum")`

  `ret=str(p.before)`

  `Minimum` の前の文字列 `p.before` を文字列型にして `ret` に入れる。

  ATSAS のプログラムではプログラムの表示は `:` で始まるのでそこで分ける。

  `m=ret.split(':')`

  `m` の中には ”\” などが入っているのでそこで区切る。

  `Ra=float(m[-1].split("\\")[0])`

crysol の対話モードの自動応答での実行

できるだけわかりやすいように、無駄な部分は省いている。

出力の文字列が乱れているが、気になるのなら `print` がある文をコメントアウトすればよい。

[1]:
```
def auto_crysol(pdb_base,dat_base):

    import pexpect


    p = pexpect.spawn("crysol")
    #2-1.  Enter your option ..................... <            0 >:
    p.expect(r">:")
    print (p.before+p.after)
    p.sendline("")
    #2-2.  Brookhaven file name ................... <         .pdb >:
    p.expect(r">:")
    print (p.before+p.after)
    p.sendline(pdb_base)
    #2-3. Maximum order of  harmonics ............ <           15 >:
    p.expect(r">:")
    print (p.before+p.after)
    p.sendline("")
    #2-4. Order of Fibonacci grid ................ <           17 >:
    p.expect(r">:")
    print (p.before+p.after)
    p.sendline("")
    #2-5. Maximum s value ........................ <        .5000 >:
    p.expect(r">:")
    print (p.before+p.after)
    p.sendline("")
    #2-6.  Number of points ...................... <           51 >:
    p.expect(r">:")
    print (p.before+p.after)
    p.sendline("201")
    #2-7.  Account for explicit hydrogens? [ Y / N ] <           No >:
    p.expect(r">:")
    print (p.before+p.after)
    p.sendline("")

    #4-1.   Fit the experimental curve [ Y / N ] .. <          Yes >:
```

(次のページに続く)

(前のページからの続き)

```
    p.expect(r">:")
    print (p.before+p.after)
    p.sendline("")
    #4-2.  Enter data file ....................... <         .dat >:
    p.expect(r">:")
    print (p.before+p.after)
    p.sendline(dat_base)
    #5-1. Subtract constant ...................... <           no >:
    p.expect(r">:")
    print (p.before+p.after)
    p.sendline("N")

    #5-2. 2 *  sin(theta)/lambda [1/nm]  (4) ..... <            1 >:
    p.expect(r">:")
    print (p.before+p.after)
    p.sendline("")

    #6. Electron density of the solvent, e/A**3  <        .3340 >:
    p.expect(r">:")
    print (p.before+p.after)
    p.sendline("")

    #7. Plot the fit [ Y / N ] ................. <          Yes >:
    p.expect(r">:")
    print (p.before+p.after)
    p.sendline("N")

    #8-1.  Another set of parameters [ Y / N ] .... <           No >:
    p.expect(r">:")
    print (p.before+p.after)
    p.sendline("Y")

    #8-2. Minimize again with new limits [ Y / N ] <           No >:
    p.expect(r">:")
    print (p.before+p.after)
    p.sendline("Y")


    p.expect(r"Minimum")
    ret=str(p.before)
    # print ('ret',ret)
    #Average atomic radius is ............................... :
    m=ret.split(':')
    # print('m',m[-1].split("\\")[0])
    Ra=float(m[-1].split("\\")[0])

    # 9-1. Minimum radius of atomic group ......... <        1.400 >:
    p.expect(r">:")
    print (p.before+p.after)
    p.sendline(str(Ra))

    # 9-2.  Maximum radius of atomic group ......... <        1.800 >:
    p.expect(r">:")
    print (p.before+p.after)
    p.sendline(str(Ra))

    # 9-3. Smax in the fitting range ............... <       0.4984 >:
    p.expect(r">:")
    print (p.before+p.after)
    p.sendline("")

    # 9-4.  Minimum contrast in the shell .......... <        0.000 >:
    p.expect(r">:")
    print (p.before+p.after)
    p.sendline("")

    # 9-5.  Maximum contrast in the shell .......... <  7.5000E-002 >:
    p.expect(r">:")
    print (p.before+p.after)
    p.sendline("")

    p.expect(r"Minimum")
    ret=str(p.before)
    # print ('ret',ret)
    #Van der Waals volume is ................................ :
    m=ret.split(':')
    # print('m',m[-1].split("\\")[0])
    ExVol=float(m[-1].split("\\")[0])

    # 9-6  Minimum excluded volume ................. <  1.6108E+004 >:
    p.expect(r">:")
    print (p.before+p.after)
    p.sendline(str(ExVol))

    # 9-7.  Maximum excluded volume ................ <  1.8720E+004 >:
    p.expect(r">:")
    print (p.before+p.after)
    p.sendline(str(ExVol))

    # 9-7. Subtract constant ...................... <           no >:
    p.expect(r">:")
    print (p.before+p.after)
```

(次のページに続く)

(前のページからの続き)

```
    p.sendline("")

    # 10-1. Plot the fit [ Y / N ] .................. <          Yes >:
    p.expect(r">:")
    print (p.before+p.after)
    p.sendline("N")

    # 10-2. Another set of parameters [ Y / N ] .... <           No >:
    p.expect(r">:")
    print (p.before+p.after)
    p.sendline("")

    #p.terminate()
    p.expect(pexpect.EOF)

    return
```

- 6lyz.pdb : 結晶構造
- lyzexp.dat : 実験データ

を入力して上の関数 auto_crysol を実行する。

```
[ ]: auto_crysol('6lyz.pdb','lyzexp.dat')
```

この実行例中の auto_crysol() を参考にして自分の使いたい **ATSAS** のプログラム用に書き換えると良いであろう。

.

3 まとめ

小角散乱のデータ解析は、

1. プログラムの開発
2. 結果の 3 次元構造の妥当性
3. それを導く元になったシミュレーションのフィットの検証

が必要となる。特に、タンパク質溶液散乱の解析プログラムにおいては **ATSAS** が標準化されており、実際、ユーザーが行う作業は **ATSAS** のプログラムのパラメータを変えて最適な解を探していく過程が主である。

本稿で紹介したビーズモデル計算は比較的単純な解析工程であるが、Rigid Body Modeling など複雑な解析工程を含む解析では、その結果がどれぐらいの robustness があるかを知ってもらうためにも解析レポートを提示することは重要だと思われる。しかし、現状では図などを準備してレポートを作成するのはかなりの作業量になってしまう。3 次元構造をその場でぐるぐるまわせて、フィットの具合も提示できる **jupyter-notebook** は、良いコミュニュケーションツールとなりうるだろう。

jupyter-notebook のベースとなっている python には、本稿で紹介しなかったが

- 数値演算 Numpy
- 科学技術計算 Scipy
- グラフ表示 matplotlib
- データ分析 pandas
- 数式処理 Sympy
- 機械学習 scikit-learn

など処理性能が高い基礎ライブラリが整っており、オープンでこれからも発展が期待できる。以前は応用数学の原理を勉強するだけでも大変だった最新の最適化ルーチンが容易に実装できたりする。それなりの学習コストはかかるが、それだけの価値はあるのでまだ使ったことのない人はぜひ検討していただきたい。

本稿で用いたファイルは、以下のとおりである。自由に変更、使用していただいて結構であるが、いかなる保証もいたしませんのであくまで自己責任ということでお願いします。

- 結晶構造より理論散乱曲線の計算
  - `crysol.ipynb`
- 理論散乱曲線から動径分布関数の計算
  - `gnom.ipynb`
- 動径分布関数からビーズ構造の計算
  - `shapes_main.ipynb`
  - `shapes_data_analysis.ipynb`
  - `shapes_main.py`
  - `shapes_module.py`
- ビーズモデルの統計処理
  - `damfilt_analysis.ipynb`
  - `beadsmodel_analysis.ipynb`
- より発展的な話題
  - `advanced.ipynb`
  - :download:‘lyzexp.dat <./M_M/advanced//lyzexp.dat> [3]

なお、 **`shapes_v13.py`** の **`jupyter-notebook`** 用の書き換えは私の研究室の鈴木日菜美さんによるものである。[9]

参考文献

[1] Hai Nguyen, David A Case, Alexander S Rose (2018) NGLview–interactive molecular graphics for Jupyter notebooks, Bioinformatics, 34, 1241–1242,

[2] Franke, D., Petoukhov, M.V., Konarev, P.V., Panjkovich, A., Tuukkanen, A., Mertens, H.D.T., Kikhney, A.G., Hajizadeh, N.R., Franklin, J.M., Jeffries, C.M. and Svergun, D.I. (2017) ATSAS 2.8: a comprehensive data analysis suite for small-angle scattering from macromolecular solutions J. Appl. Cryst. 50, 1212-1225

[3] Svergun D.I., Barberato C. and Koch M.H.J. (1995) CRYSOL - a Program to Evaluate X-ray Solution Scattering of Biological Macromolecules from Atomic Coordinates J. Appl. Cryst. , 28, 768-773.

[4] Badger, J. (2019). A new algorithm for the reconstruction of protein molecular envelopes from X-ray solution scattering data J. Appl. Cryst. 52, 937-944.

[5] Svergun, D. I. (1992). Determination of the regularization parameter in indirect-transform methods using perceptual criteria J. Appl. Cryst. 25, 495-503.

[6] V.V. Volkov and D.I. Svergun (2003). Uniqueness of ab-initio shape determination in small-angle scattering. J. Appl. Cryst. 36, 860-864.

[7] Anne T. Tuukkanen, Gerard J. Kleywegt and Dmitri I. Svergun(2016) Resolution of ab initio shapes determined from small-angle scattering IUCrJ. 3, 440-447.

[8] P.V.Konarev, M.V.Petoukhov and D.I.Svergun (2016) Rapid automated superposition of shapes and macromolecular models using spherical harmonics. J Appl Cryst. 49, 953-960.

[9] 鈴木日菜美　(2019) 2019 年度岐阜大学工学部　卒業論文