Protein Superpositioning using Bayesian Inference

by Basile Rommes
Supervisor Thomas Hamelryck

March 1, 2019

1 Introduction

One possible Bayesian Interference model for protein superpositioning is the Gaussian perturbation model found in the expectation maximisation (EM) approach used in the paper by Theobald et al. [1] and implemented in the program 'Theseus'. Here, the different protein structures X_i are represented as deviations from a mean structure M. The mean structure M is subject to some Gaussian noise E_i (due to the central limit theorem), specific to each protein X_i, as well as a rotation R_i and a translation T_i.

$$X_i = R_i \cdot (M + E_i) + T_i$$

If we select M such that it has the same orientation as one of our proteins X_i, we can omit the rotation for one of the structures. When superpositioning only two protein structures, the model takes the form:

$$X_1 = (M + E_1) + T_1$$
$$X_2 = R \cdot (M + E_2) + T_2$$

Thus X_1 and X_2 are multidimensional and normally distributed ($\mathcal{N}_{3n}(\mu, \Sigma)$) with the parameters:

$$X_1 \sim \mathcal{N}_{3n}(M + T1, V \otimes U)$$
$$X_2 \sim \mathcal{N}_{3n}(RM + T2, V \otimes U)$$

with covariance matrices $V := I_3$ and $U := \sigma^2 I_n$ from $\mathcal{MN}_{n,3}(M, U = \sigma^2 I_n, V = I_3)$ (see [1] end of 3.2), where n is the number of $C_\alpha s$ in X_i and $\otimes$ is the Kronecker product.
(See chapter 3 for a derivation from the matrix normal to the multidimensional normal distribution.)
The probability of seeing X_1 and X_2 under the Gaussian perturbation model is:

$$P(X_1 | M, I_3 \otimes \sigma^2 I_3)$$

$$P(X_2|M, I_3 \otimes \sigma^2 I_3)$$

Hereinafter, I shall explain the choice of the priors, which is crucial to the design of the Bayesian Interference model. Moreover, to get from the model to a log-likelihood posterior distribution, we sample from a matrix normal distribution, as described in section 5.

2 Reading in real protein data and describing it according to the model

Our protein of choice is 2SDF, read-in as a pdb file downloaded from RCSB PDB. From the structure, n C_α coordinates are extracted:

```
parser = PDB.PDBParser(QUIET=True)
structure = parser.get_structure("1ENH", "1enh.pdb")

# cas from loaded protein
cas_real = []

for model in structure:
    for chain in model:
        for residue in chain:
                try:
                    # creates a list of lists containing all residue atom coordinates
                    cas_real.append(residue['CA'].get_coord())
                except:
                    pass
```

The coordinates are put into matrix form and centered around the origin.

```
# Vertical stack to concatenate list of lists of coordinates into np array
cas_real_array = np.vstack(cas_real)
print(f"cas_real_array:\n{cas_real_array}")
cas_real_array_centered = center(cas_real_array)
```

3 Prior over rotations R_i

Because new random variables have to be sampled from the uniform distribution everytime we define a rotation (see further down how to get from the RV to the rotation), this prior is defined in a python function. Since the random variables have to be saved as symbolic variables with unique names in the pymc3 model, the function sample_R takes an integer used in the variable name as argument. Note however, that for this simple case, we only need one rotation, and therefore only call sample_R once.

```
112         """sample a unit quaternion and transform it into a rotation
    matrix"""
113         # the first argument states that i will be the name of the
    rotation made
114         ri_vec = pm.Uniform('r' + str(i), 0, 1, testval=0.5, shape=3)  #
    Note shape is 3-Dimensional
115         theta1 = 2 * np.pi * ri_vec[1]
```

The prior distribution for the rotations R_i was chosen from a uniform distribution of unit quaternions. This method is presented in a book by Xavier Perez-Sala et al.[2].
Let $\mathcal{U}(.)$ be the uniform distribution and let $q = (w, x, y, z)$ be a quaternion of the form w, xi, yj, zk representing a rotation in 3D-space.
Xavier Perez-Sala et al. notes that a uniformly distributed element of a complete group (e.g. group of quaternions $q = (w, x, y, z)$) can be achieved by multiplication of a uniformly distributed element from a subgroup (e.g. the group of planar rotations around the Z-axis: $q = (c, 0, 0, s)$) with a uniformly distributed coset (e.g. rotations of the Z-axis: $q = (w, x, y, 0)$).

$$\mathcal{U}([c, 0, 0, s])\mathcal{U}([w, x, y, 0]) = \mathcal{U}([cw, cx + sy, -sx + cy, sw])$$

We get the quaternion by sampling the parameters θ_1, θ_2, r_1 and r_2 and plugin them into this expression:

$$q = (w, x, y, z) = (r_2 \cos\theta_2, r1 \sin\theta_1, r_1 \cos\theta_1, r_2 \sin\theta_2)$$

Let's have a look at how this is done in theano:

```
120         qx = r1 * tt.sin(theta1)
121         qy = r1 * tt.cos(theta1)
122         qz = r2 * tt.sin(theta2)
```

Three random variables r_0, r_1, r_2 are sampled in the code:

```
111         ri_vec = pm.Uniform('r' + str(i), 0, 1, testval=0.5, shape=3)  #
    Note shape is 3-Dimensional
```

Where r_1 and r_2 define the θ angles:

```
111         theta1 = 2 * np.pi * ri_vec[1]
112         theta2 = 2 * np.pi * ri_vec[2]
```

While r_0 defines the two radii r_1 and r_2:

```
115         r1 = tt.sqrt(1 - ri_vec[0])
116         r2 = tt.sqrt(ri_vec[0])
```

The relation between quaternion $q = (w, x, y, z)$ and a 3×3 rotation matrix is (see Coutsias et al 2004 [3])

$$(w, x, y, z) = \begin{bmatrix} w^2 + x^2 - y^2 - z^2 & 2(xy - wz) & 2(xz + wy) \\ 2(xy + wz) & w^2 - x^2 + y^2 - z^2 & 2(yz - wx) \\ 2(xz - wy) & 2(yz + wx) & w^2 - x^2 - y^2 + z^2 \end{bmatrix}$$

```
126         R = t.shared(np.eye(3))
127 
128         # filling the rotation matrix
129         # Evangelos A. Coutsias, et al "Using quaternions to calculate
    RMSD" In: Journal of Computational Chemistry 25.15 (2004)
130 
131         # Row one
132         # tt.sqr = square, tt.sqrt = square-root
133         R = tt.set_subtensor(R[0, 0], qw**2 + qx**2 - qy**2 - qz**2)
134         R = tt.set_subtensor(R[0, 1], 2*(qx*qy - qw*qz))
135         R = tt.set_subtensor(R[0, 2], 2*(qx*qz + qw*qy))
136         # R = tt.set_subtensor(R[0, :], [a, b, c])
137 
138         # Row two
139         R = tt.set_subtensor(R[1, 0], 2*(qx*qy + qw*qz))
140         R = tt.set_subtensor(R[1, 1], qw**2 - qx**2 + qy**2 - qz**2)
141         R = tt.set_subtensor(R[1, 2], 2*(qy*qz - qw*qx))
142 
143         # Row three
144         R = tt.set_subtensor(R[2, 0], 2*(qx*qz - qw*qy))
145         R = tt.set_subtensor(R[2, 1], 2*(qy*qz + qw*qx))
146         R = tt.set_subtensor(R[2, 2], qw**2 - qx**2 - qy**2 + qz**2)
```

This rotation is an uninformative prior, i.e. it is chosen for the sole reason of weighing all rotations equally likely (uniform distribution), without introducing any bias.

4 Prior mean structure M

The mean structure is modelled as a chain of C_α-atoms. This protein backbone can be conceptionalized as a sequence of vectors, representing the direct transitions from one C_α position to the next.

We can represent those vectors via polar-coordinates (r, θ, ϕ) where r is the vector's length, and since all vectors are of same length, r is constant and can be interpreted as the radius of a sphere. θ is the azimuthal angle and ϕ the polar angle on this sphere (see the online article by Cory Simon[4]).

The relationship between cartesian (x, y, z) coordinates and polar-coordinates (r, θ, ϕ) is:

$$x = r \cdot cos(\theta) sin(\phi)$$

$$y = r \cdot sin(\theta) sin(\phi)$$

$$z = r \cdot cos(\phi)$$

In theano:

```
    x = tt.sin(phi) * tt.cos(theta)
    y = tt.sin(phi) * tt.sin(theta)
    z = tt.cos(phi)
```

As Cory Simon explains, due to the way a sphere's surface area in polar-coordinates is calculated, sampling θ and ϕ from a uniform distribution doesn't result in uniformly distributed points on a sphere. I.e. vectors from origin to sphere surface, that are sampled by this method will not be uniformly distributed.
The first approach gives uniform points on a $[0, 2\pi[\times[0, \pi[$ square (with θ and ϕ the respective coordinates) but achieving uniform distribution on a sphere requires a different approach. The detailed derivation can be read in [4], but it boils down to the formula:

$$\mathbb{P}\{F^{-1}(U) \leq x\} = F(\phi)$$

Where U is the uniform distribution and F is the cumulative distribution function.

This makes the algorithm for sampling the distribution $f(\phi)$ via inverse transformation sampling:

- Sample u from $U(0, 1)$.
- Compute $\phi = F^{-1}(u)$, in the 2-sphere case: $F^{-1}(u) = arccos(1 - 2u)$.
- treat ϕ as a random number drawn from the distribution $f(\phi)$

Meanwhile θ is calculated straightforward via $2\pi x_1$ where x_1 is uniformly sampled.

```
    x1 = pm.Uniform("x1", 0, 1, shape=n)
    theta = 2 * np.pi * x1
```

For this simple prior, we are going to assume equal distances of $3.8\mathring{A}$ between neighbouring C_α's.

```
    cas = pm.Deterministic("cas", 3.8*tt.stack([x, y, z], axis=1))
```

In order to get the C_α trace - a connected chain of the C_α atoms - we need to sum over all unit vectors. The resulting trace is then centered to the origin by substracting it's center of mass.

```
    cas_trace = tt.extra_ops.cumsum(cas, axis=0)
```

```
    M = cas_trace-tt.mean(cas_trace, axis=0)
```

5 Gaussian Prior

The protein structures consisting out of p atoms are represented in our model as $n \times 3$ matrices X_i (just as in Theobald et al. [1]). To model Gaussian noise around the mean, we sample from the matrix normal distribution. A multivariate normal distribution $\mathcal{N}_k(\mu, \Sigma)$ gives the probability density of k independent gaussian distributed random variables $X_1, .., X_k$ with mean vector μ and covariance matrix Σ. The matrix normal distribution $\mathcal{MN}_{n,p}(M, U, V)$ is a generalization of the multivariate normal distribution to matrix-valued random variables, where:

- M is the $n \times p$ mean matrix
- U is the $n \times n$ covariance among rows matrix, in our case that is the covariance among atom-positions
- V is the $p \times p$ covariance among columns matrix, in our case a 3×3 matrix with covariance among x-, y- and z-coordinates of atoms

Since version 3.3 pyMC3 implements the matrix normal via the function *MatrixNormal*.

```
224     X1 = pm.MatrixNormal("M_E1_T1", mu=M_T1, rowcov=U, colcov=V, shape=(n,
        3), observed=cas_real_array_centered)
225     X2 = pm.MatrixNormal("M_E2_T2", mu=M_R2_T2, rowcov=U, colcov=V, shape
       =(n, 3), observed=cas_real_array_centered)
```

6 Calculate the prior over translations T_i

The translations are vectors in $\mathbb{R}^3$ that are drawn from a normal distribution. To have equal probability of all directions a mean of $\mu = 0$ is chosen. The standard deviation is tweaked to achieve optimal results, see the results section 8.2.

```
100     T2 = pm.Normal('T2', mu=0, sd=0.1, shape=3)
```

7 Maximum a posteriori estimate

The MAP estimation on our model is called, with the number of iterations capped by a maximum of 10000.

```
287 map_estimate = pm.find_MAP(maxeval=10000, model=inner_model)
```

8 Results

8.1 Validating the model on 1ENH

A sample of 10 consecutive C_α atoms from the protein 1ENH was used to test the models ability to approach a given structure. Figure 1 shows that the short C_α trace was approximated well, with an RMSD around 0.0207 Å.

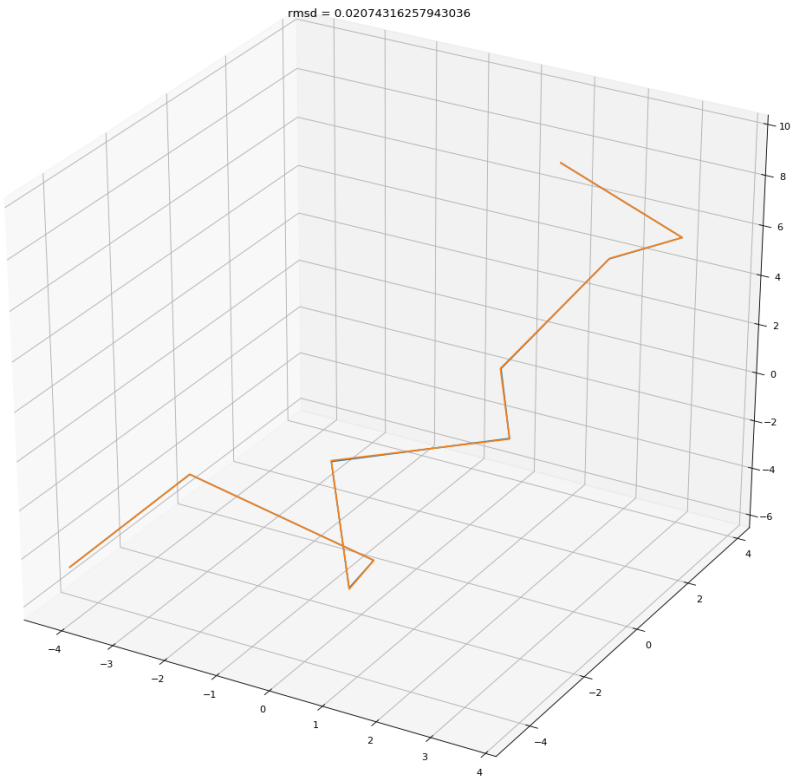

Figure 1: C_α trace sampled from protein 1ENH (blue) and the transformed mean structure calculated based on the model (orange). A standard deviation of 1 was used for sampling of translation and rotation parameters.

Even when modelling all 54 C_αs from 1ENH, the approximation performs fairly good with an RMSD of approx. 0.663 Å(see figure 2).

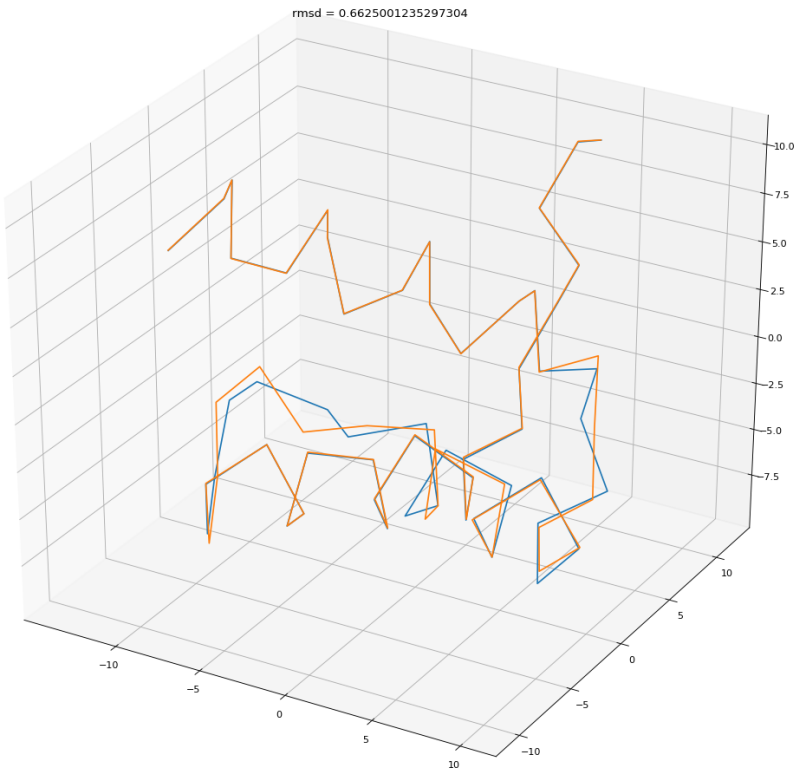

Figure 2: Entire C_α trace of protein 1ENH (blue) and the transformed mean structure calculated based on the model (orange). A standard deviation of 1 was used for sampling of translation and rotation parameters.

8.2 Comparison of conformations of 2SDF

The pdb file for 2SDF consists of 30 states. Each of them feature a small α-helix and three β-sheets that stay almost completely rigid in all 30 conformational states, as do the loops in-between them. The polypeptide chains at both ends of the protein however, feature a high degree of conformational variability.

The proteins states have previously used for probabilistic superpositioning methods in a paper by Theobald et al. [5], see figure 3, which compares the results of a maximum likelihood (ML) and a simple least squares (LS) approach to the known mean structure.

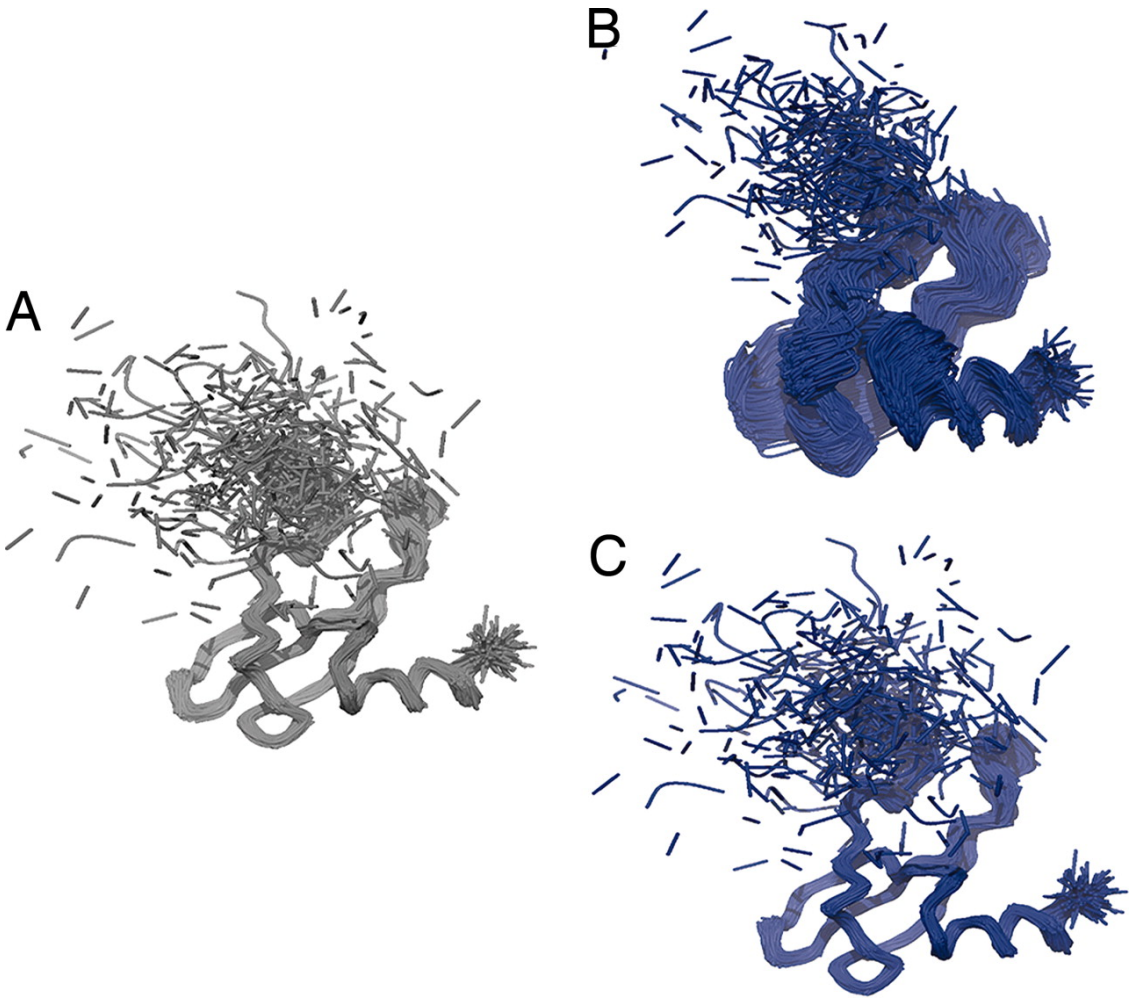

Figure 3: from [5]: (A) The true superposition, generated from a known mean structure with known covariance matrices, before arbitrary translations and rotations have been applied. (B) An ordinary LS superposition of the simulated data. (C) A ML superposition of the simulated data, assuming a diagonal landmark covariance matrix U (i.e., no correlations), dimensional covariance matrix $V = I$, and inverse gamma distributed variances.

Figure 4: State 1 of 2SDF

In the following the performance of the model is evaluated on two conformations of a protein. Running the model on 2SDF state 1 and state 2 (according to the enumeration of the pdb file) provides us with a mean structure of the two. Figure 5 visualises the model output for a standard deviation of 1 for both the translation and the gaussian prior.

The python script also calculates the RMSD between the computed states in relation to the mean as well as the RMSD in-between both states 1.

Standard Deviation of 1

Table 1: RMSDs, std = 1

	state1	state2	mean
state1	0	6.689859433625074	3.4552185971651443
state2	6.689859433625074	0	3.4315393540147574
mean	3.4552185971651443	3.4315393540147574	0

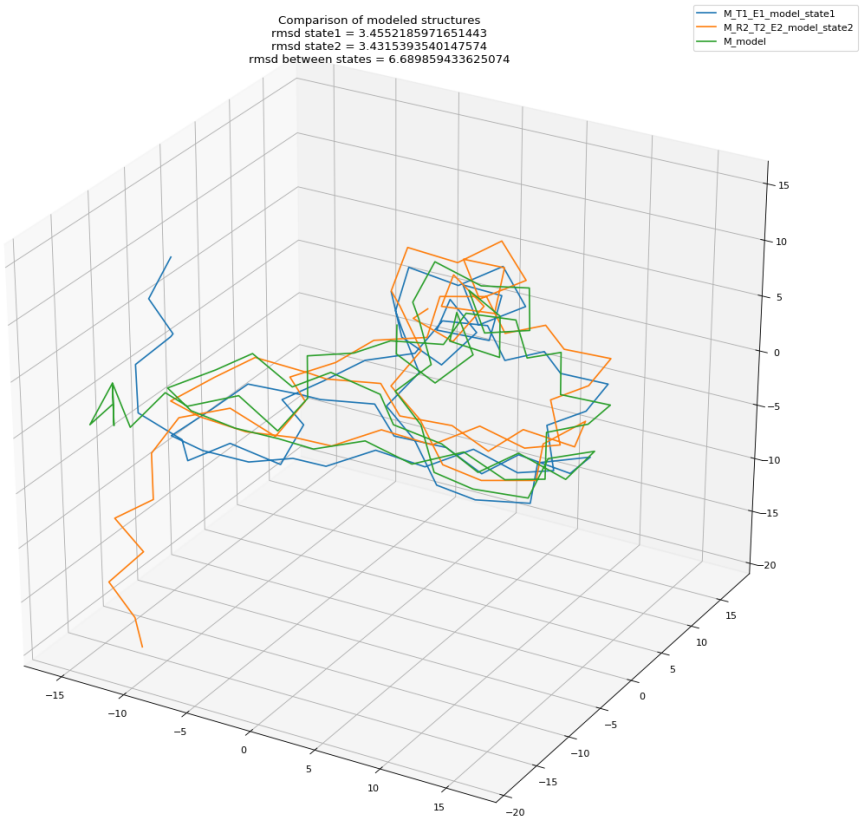

Figure 5: Comparison of C_α traces of state 1 (blue) and state 2 (orange) of 2SDF. The mean structure computed by the model is shown in green. $sd = 1$

8.3 Tweaking model parameters for 2SDF

The model remains robust within a certain value range for the standard deviation of the mentioned priors. Below are the results for given standard variations. Each time, the same value was used for the priors of T1, T2, E1 and E2.

Standard Deviation of 1e-6

Table 2: RMSDs, std = 1e-6

	state1	state2	mean
state1	0	6.689859433625074	3.9544828207178697
state2	6.689859433625074	0	3.952038783096669
mean	3.9544828207178697	3.952038783096669	0

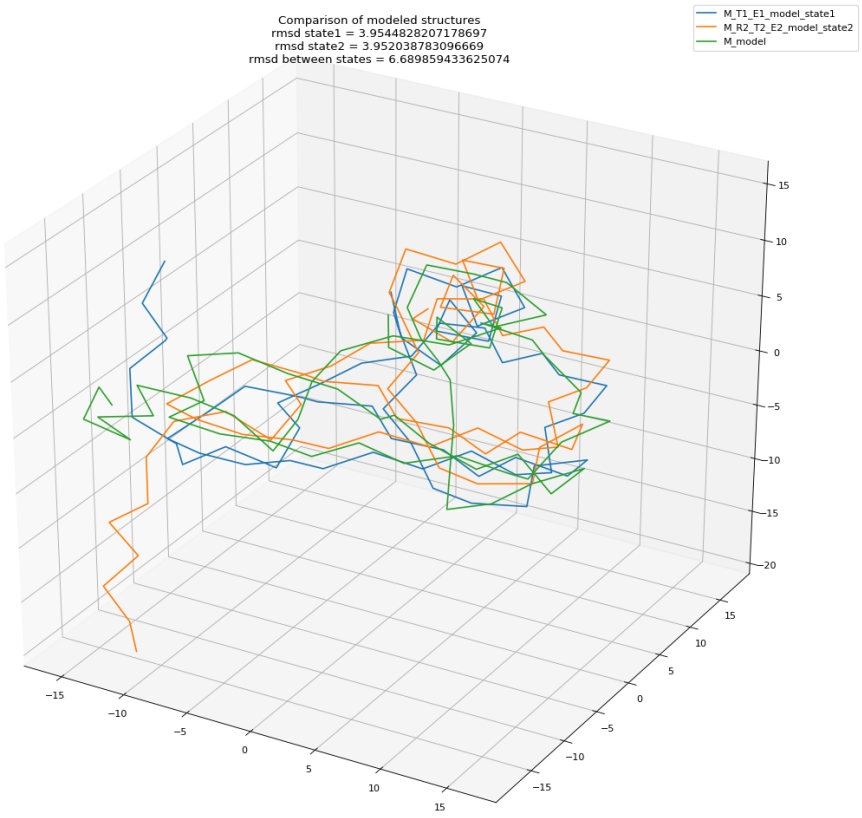

Figure 6: Comparison of C_α traces of state 1 (blue) and state 2 (orange) of 2SDF. The mean structure computed by the model is shown in green. $sd = 1e-6$

Standard Deviation of 1e-8

If we keep decreasing the standard deviation, the mean structure will lose all similarity to the models, as seen in figure 3.

Table 3: RMSDs, std = 1e-8

	state1	state2	mean
state1	0	6.689859433625074	67.84120706803758
state2	6.689859433625074	0	67.45500802514539
mean	67.84120706803758	67.45500802514539	0

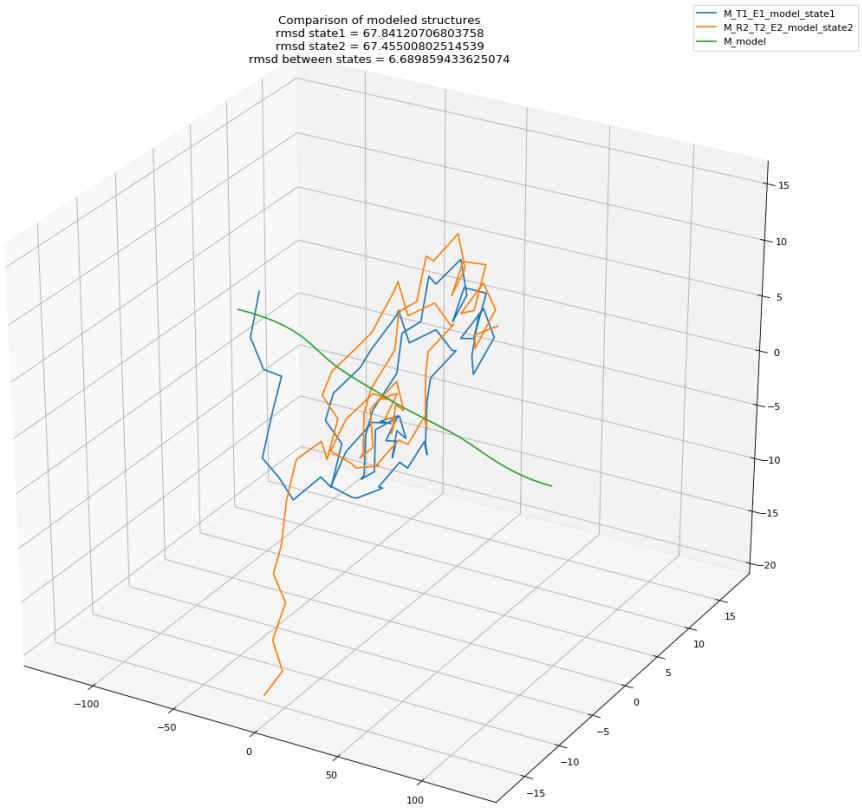

Figure 7: Comparison of C_α traces of state 1 (blue) and state 2 (orange) of 2SDF. The mean structure computed by the model is shown in green. $sd = 1e-8$

Standard Deviation of 10^8

As this example shows, even high standard deviations up to $1e8$ work well on the model.

Table 4: RMSDs, std = 1e8

	state1	state2	mean
state1	0	6.689859433625074	3.4114687224708593
state2	6.689859433625074	0	3.4104725690023194
mean	3.4114687224708593	3.4104725690023194	0

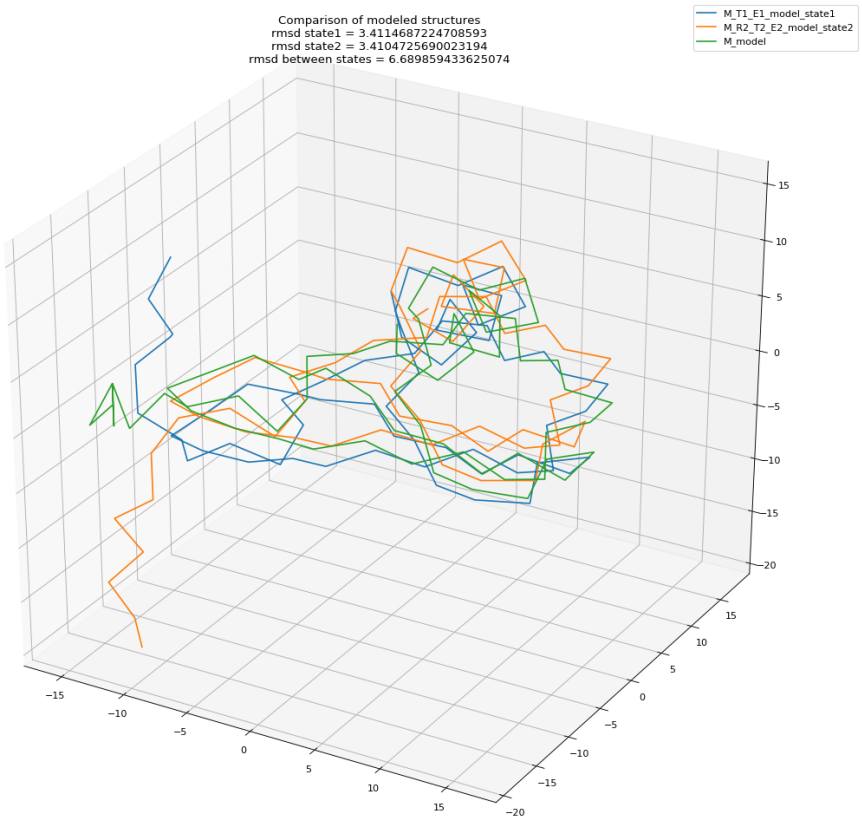

Figure 8: Comparison of C_α traces of state 1 (blue) and state 2 (orange) of 2SDF. The mean structure computed by the model is shown in green. $sd = 1e8$

While tweaking the standard variance modifies the mean structure, it has no effect on the models for state 1 and state 2 of 2SDF. Ideally we would want to tweak the model such that middle parts of the traces are a tight fit, since they have a low variability from state to state - while allowing for a high divergence at both ends of the traces.

8.4 Outlook

- the sampling of the alpha carbon trace from random vectors doesn't take legal C_α-C_α angles into account. Thus we might sample two consecutive vectors in a strongly acute angle, which wouldn't be able to form naturally due to steric effects. However, the effect of ignoring this constraint is expected to be negligible for the end-result.

- Due to the complex nature of the sampling of the mean prior and the rotation prior,

adjusting their parameters (mean, standard deviation) is non-trivial. The model might be improved by experimenting with parameter variations of these priors.

- Modelling and comparing more states simultaneously (e.g. all 30 states of 2SDF) and making sausage plots such as in Figure 3 helps giving a more overall picture of the bayesian inference model's success. Such a plot would also lend itself for comparison to the Maximum likelihood approach used by Theobald et al.
- use automatic differentiation variational inference (ADVI) instead of MAP for parameter inference

8.5 Acknowledgment

This work is based on a project done by William Paul Bullock under the supervision of Thomas Hamelryck [6].

References

[1] Douglas L. Theobald and Phillip A. Steindel. "Optimal simultaneous superpositioning of multiple structures with missing data". In: *Bioinformatics* 28.15 (2012), pp. 1972–1979. ISSN: 13674803. DOI: 10.1093/bioinformatics/bts243.

[2] Xavier Perez-Sala et al. *Uniform Sampling of Rotations for Discrete and Continuous Learning of 2D Shape Models*. IGI Global, Jan. 1970.

[3] Evangelos A. Coutsias, Chaok Seok, and Ken A. Dill. "Using quaternions to calculate RMSD". In: *Journal of Computational Chemistry* 25.15 (2004), pp. 1849–1857. ISSN: 01928651. DOI: 10.1002/jcc.20110. arXiv: arXiv:1011.1669v3.

[4] *Article by Cory Simon*. URL: http://corysimon.github.io/articles/uniformdistn-on-sphere/ (visited on Feb. 23, 2018).

[5] D. L. Theobald and D. S. Wuttke. "Empirical Bayes hierarchical models for regularizing maximum likelihood estimation in the matrix Gaussian Procrustes problem". In: *Proc. Natl. Acad. Sci. U.S.A.* 103.49 (Dec. 2006), pp. 18521–18527.

[6] William Paul Bullock. *Probabilistic Programming for Protein Superpositioning*.