"Nice" plotting of proteins: I

A widely used display of protein shapes is based on the coordinates of the alpha carbons -- C_α -s. The coordinates of the C_α -s are connected by a continuous curve that roughly follows the direction in space of the protein chain.

Each amino acid has only one alpha carbon and the distance between sequential C_α -s is about 3.8 angstrom. There is only one exception to the 3.8 "rule", which is the cis isomer of proline for which the distance is smaller than the above.

The uniform distribution of the C_α -s along the protein curve makes plots of the protein backbone relatively easy to do. The simplest solution (that we used already) is to connect the coordinates by a straight line interpolating from one C_α to the next. This procedure creates a zigzag line, which is ok, but not great and not pleasing to the eye. "Staring" at proteins shapes and looking for interesting features can be difficult (examples for interesting features are structural domains that are shared between the proteins, similar active sites of evolutionary related proteins, etc). There are many automated algorithms to look for the features mentioned above. However, the "human eye" is in many cases a better detection device.

Therefore, producing better pictures of protein chains is likely to help basic research in this area. One thing that our eyes "do not like" is discontinuous derivatives. The eye can detect second order derivative that is not continuous. The curve of the protein chain, which we plot in three dimensions by connecting linearly the C_α -s, is discontinuous in the first derivative. Here we seek another representation that will make the protein curve differentiable at least to a second order. That is, we are facing with the topic of interpolation between points (the C_α -s positions) that represent a curve.

Interpolation

The first approach to interpolation that we consider is the use of polynomials. A set of points $\{(x_i y_i)\}_{i=1}^{N}$ can be approximated by a polynomial y and a variable x

$$y = c_0 + c_1 \cdot x + c_2 \cdot x^2 + c_3 \cdot x^3 + \ldots + c_n x^n$$

An alternative representation of the $n-th$ order polynomial is

$$y = \left(c_0 + \left(c_1 + \left(c_2 + (\ldots)\right)x\right)x\right)$$

or yet another representation using the roots of the polynomial

$$y = c_n (x - r_1) \ldots (x - r_n)$$

(Note that in creating a protein curve we need to consider three functions. A function for each of the coordinates $x, y,$ and z. The curve will be parameterized with independent continuous variable t that is equal to the amino acid index i at the coordinates of $C_\alpha(i)$. Hence, we compute the continuous curves $x(t), y(t), z(t)$. The polynomial is continuous and differentiable at all orders and therefore suggests itself as a plausible representation of the protein curve).

In MATLAB we can obtain the roots of the polynomial by the "roots" command. For example, consider the polynomial below:

$$y = 5 \cdot y^3 + y^2 + 2$$

The "roots" command finds the zeroes of the polynomial

```
>> roots([5 1 0 2])

ans =

  -0.8099
   0.3049 + 0.6332i
   0.3049 - 0.6332i
```

The roots determine the polynomial up to a constant multiplier. The command "poly" creates the polynomial from the root:

```
>> poly(roots([5 1 0 2]))

ans =

   1.0000    0.2000    0.0000    0.4000
```

The polynomial so created (always) has a coefficient $c_n = 1$, to recover the original polynomial we need to multiply all coefficients by the original c_n, and by 5 in the specific example above.

How to compute polynomials efficiently?

Here is a MATLAB loop that computes $y = c_0 + c_1 \cdot x + c_2 \cdot x^2 + c_3 \cdot x^3 + ... + c_n x^n$
The coefficients are stored in a vector c of length $n+1$

```
len= length(c);
polynomial = c(1);
ym=1;
for i=2:len
```

```
        ym = y * ym;
        polynomial = polynomial + c(i)*ym;
end
```

The number of operations to compute the value of the polynomial is (len-1)*3

It is possible to compute the polynomial more efficiently by using the equivalent formula $y = \left(c_0 + \left(c_1 + \left(c_2 + (...)\right)x\right)x\right)$

We have:

```
len= length(c);
polynomial = c(len);
for i=len-1:-1:1
        polynomial = polynomial*y + c(i);
end
```

Only (len-1)*2 operations are required this time, so this is clearly a better way of computing values of the polynomial.

Yet another formulation (Newton representation) is given below. Consider the interpolation between four points $\left\{\left(x_i, y_i\right)\right\}_{i=1}^{4}$. In the Newton representation we write the polynomial as:

$$y = c_1 + c_2\left(x - x_1\right) + c_3\left(x - x_1\right)\left(x - x_2\right) + c_4\left(x - x_1\right)\left(x - x_2\right)\left(x - x_3\right)$$

To determine the coefficients c_i we use the four points

$$\begin{aligned}
y_1 &= c_1 \\
y_2 &= c_1 + c_2\left(x_2 - x_1\right) \\
y_3 &= c_1 + c_2\left(x_3 - x_1\right) + c_3\left(x_3 - x_1\right)\left(x_3 - x_2\right) \\
y_4 &= c_1 + c_2\left(x_4 - x_1\right) + c_3\left(x_4 - x_1\right)\left(x_4 - x_2\right) + c_4\left(x_4 - x_1\right)\left(x_4 - x_2\right)\left(x_4 - x_3\right)
\end{aligned}$$

This set of linear equations (for the c_i) is relatively easy to solve. In a matrix form we write:

$$\begin{bmatrix}
1 & 0 & 0 & 0 \\
1 & \left(x_2 - x_1\right) & 0 & 0 \\
1 & \left(x_3 - x_1\right) & \left(x_3 - x_1\right)\left(x_3 - x_2\right) & 0 \\
1 & \left(x_4 - x_1\right) & \left(x_4 - x_1\right)\left(x_4 - x_2\right) & \left(x_4 - x_1\right)\left(x_4 - x_2\right)\left(x_4 - x_3\right)
\end{bmatrix}
\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} =
\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

It is obvious that $c_1 = y_1$, substituting we obtain the following linear equation

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & (x_2-x_1) & 0 & 0 \\ 0 & (x_3-x_1) & (x_3-x_1)(x_3-x_2) & 0 \\ 0 & (x_4-x_1) & (x_4-x_1)(x_4-x_2) & (x_4-x_1)(x_4-x_2)(x_4-x_3) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2-y_1 \\ y_3-y_1 \\ y_4-y_1 \end{bmatrix}$$

By dividing line 2,3, and 4 by (x_2-x_1), (x_3-x_1) and (x_4-x_1) respectively, we obtain

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & (x_3-x_2) & 0 \\ 0 & 1 & (x_4-x_2) & (x_4-x_2)(x_4-x_3) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} y_{11} \\ y_{21} \\ y_{31} \\ y_{41} \end{bmatrix}$$

where $y_{ij} = \dfrac{y_i - y_j}{x_i - x_j}$. The last matrix equation provides an immediate solution for the coefficient c_2. The same type of process can be repeated to determine other coefficients.

This brings us to a large set of problems of solving linear equalities of the type $Ax = b$ where x is a vector of unknown of length n, b is a vector of parameters of the same length and A is an $n \times n$ matrix. A simple case to star with is of triangular matrices (this includes the case which we just studied).

Triangular problems

Example

$$\begin{pmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

which is rather easy to solve. We can immediately write $x_1 = b_1 \cdot a_{11}$. Using the (now) known value of x_1 we can write for x_2, $x_2 = (b_2 - a_{21} \cdot x_1) / a_{22}$. Similarly we can write for x_3, $x_3 = (b_3 - a_{31} \cdot x_1 - a_{32} \cdot x_2) / a_{33}$

For the general case we can write an implicit solution (in terms of the "earlier" $x_j \quad j = 1, ..., i-1$)

$$x_i = \left(b_i - \sum_{j=1}^{i-1} a_{ij} \cdot x_j \right) \Big/ a_{ii}$$

Note that a similar procedure applied to the upper triangular matrix

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

To solve a general linear problem we search for a way of transforming the matrix to a triangular form (which we know already how to solve). Formally, we seek the so-called LU decomposition in which the general A matrix is decomposed into a lower triangular matrix L, and an upper triangular matrix U ($A = LU$). Note that if such a decomposition is known, we can solved the linear problem in two steps

Step 1.

$Ax = b$

$LUx = b$

$L(Ux) = b$

$Ly = b$ find y using the lower triangular matrix L

Step 2.

$Ux = y$ find x using the upper triangular matrix U

A way of implementing the above idea in practice is using Gaussian elimination. Gaussian elimination is an action that leads to a LU decomposition discussed above, even if the analogy is not obvious. In this course we will not prove the equivalence.

<u>Gaussian elimination</u>

Consider the following system of linear equations

$$\begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{pmatrix}\begin{pmatrix} x \\ x \\ x \\ x \\ x \end{pmatrix} = \begin{pmatrix} x \\ x \\ x \\ x \\ x \end{pmatrix}$$

where x denotes any number different from zero.

We can eliminate the unknown x_1 from rows 2 to n (the general matrix is of size $n \times n$). We multiply the first row by a_{i1}/a_{11} and subtract the result from row i. By repeating the process $n-1$ times we obtain the following (adjusted) set of linear equations that has the same solution

$$\begin{pmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{pmatrix}\begin{pmatrix} x \\ x \\ x \\ x \\ x \end{pmatrix} = \begin{pmatrix} x \\ x \\ x \\ x \\ x \end{pmatrix}$$

We can work on the newly obtained matrix in a similar way to eliminate x_2 from row3 3 to n. This we do by multiplying the second row by a_{i2}/a_{22} and subtract the results from rows 3 to n. The (yet another) new matrix and linear equations will be of the form

$$\begin{pmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & x & x & x \end{pmatrix} \begin{pmatrix} x \\ x \\ x \\ x \\ x \end{pmatrix} = \begin{pmatrix} x \\ x \\ x \\ x \\ x \end{pmatrix}$$

It should be obvious how to proceed with the elimination and to create (an upper) triangular matrix that we know by now how to solve.