ClassTor.py

ClassTor 1.0

Classification of **Tor**sion angles

Documentation

March 2022

Maximilian Meixner

Herein we present ClassTor, a python script for classification of ligand conformations. ClassTor separates different conformations of a molecule into classes based on the distribution of their torsion angle values. The intuitive classification of ClassTor performs a general conformational analysis of sampled space, which can be further reduced by extracting a subset of highly diverse conformations. This documentation provides information about the algorithm behind ClassTor.

Table of Content

General structure of ClassTor

ClassTor consists of an executable python script (ClassTor.py) and two python modules (classtor_spectrum.py, classtor_classification.py). While the analysis script handles command line input and writes the corresponding output files, the necessary steps for the actual classification are calculated with functions of the module files. Shortly, the algorithm behind ClassTor‘s classification consists of two main steps:

1. Individual analysis of the dihedral angle distribution of each torsion, thereby defining bins by automatically dividing the total range of torsion angle values [-180°, +180°] according to minima and maxima of this distribution (classtor_spectrum.py).

2. For every structure (frame), the combination of all torsion angle values of specified torsions is translated to the corresponding bin labels of each torsion according to the previous definition, obtaining a characteristic identifier. Subsequently, all frames with the same identifier are grouped into the same class (classtor_classification.py).

In the following, the individual steps of this classification algorithm are explained in more detail: first, generation and analysis of a distribution spectrum for each torsion and, second, the final classification considering all specified torsions. ClassTor provides a reasonable classification scheme since the definite class labels are based on the intuitive analysis of the sampled distributions of each torsion angle. Additionally, a scheme for extracting a conformationally diverse subset of centroid structures from the classification result is introduced and explained herein. The so called perturbational approach of ClassTor considers the structural information encoded in the classifier to select significantly different structures.

After elucidating the methods behind ClassTor, the command line inputs of the executing ClassTor analysis script (ClassTor.py) are summarized.

Torsional distribution spectrum (classtor_spectrum.py)

In this section the first step of the ClassTor method, the generation and characterization of a torsional distribution spectrum, is explained, which represents a pre-requisite for the subsequent classification procedure as each torsion angle range will be divided into characteristic bins. Thus, ClassTor performs this step for all (*AngNum*) torsions individually, even if a subset of torsion is specified for classification *via* the *-f* flag (see **Usage of ClassTor.py**). Additionally, after all spectra were successfully generated and characterized, ClassTor performs an internal flexibility analysis and ranks all torsions according to a flexibility score (FlexScore).

Generating the distribution spectrum

In this step the torsion angle values in the range of -180° and +180° of all torsions are analyzed individually and a distribution spectrum for each torsion is generated with the module classtor_spectrum.py. Therefore, ClassTor loops over every torsion and saves a corresponding spectrum as an output file (e.g., *spectrum_a.dat* for the torsion with label "*a*").

In more detail, the frequency of each integer value in the spectrum between -180 and +180 is determined by counting their occurrences among the provided values (black in **Figure 1**). The resulting frequencies are iteratively smoothed with a standard gaussian kernel (**Equation 1**) yielding a spectrum-like shape which represents the distribution of sampled torsion angle values,

$$kernel = exp\left(-\frac{(a-b)^2}{2\sigma^2}\right) \tag{1}$$

where *b* is the index of the current frequency and *a* is an array of all indices. The width of the gaussian kernel is given by its standard deviation *σ* (**Equation 2**),

$$\sigma = \frac{GaussKern}{\sqrt{8\log 2}} \tag{2}$$

used as the expression of the full width at half maximum. Thus, altering the *GaussKern* value will modify the width of each gaussian kernel and, ultimately, the shape of the resulting spectrum. It is important to note that for successful characterization (see below), a smooth shape of the spectrum is required. As shown in **Figure 1** smaller values of *GaussKern* produce a sharp distribution spectrum, while greater values will broaden the peaks resulting in a coarse spectrum. Consequently, in the latter case less distinct extrema are flattened and might not be detected during characterization, exemplary shown for the local minimum around -45°, which will affect the subsequent division into bins.

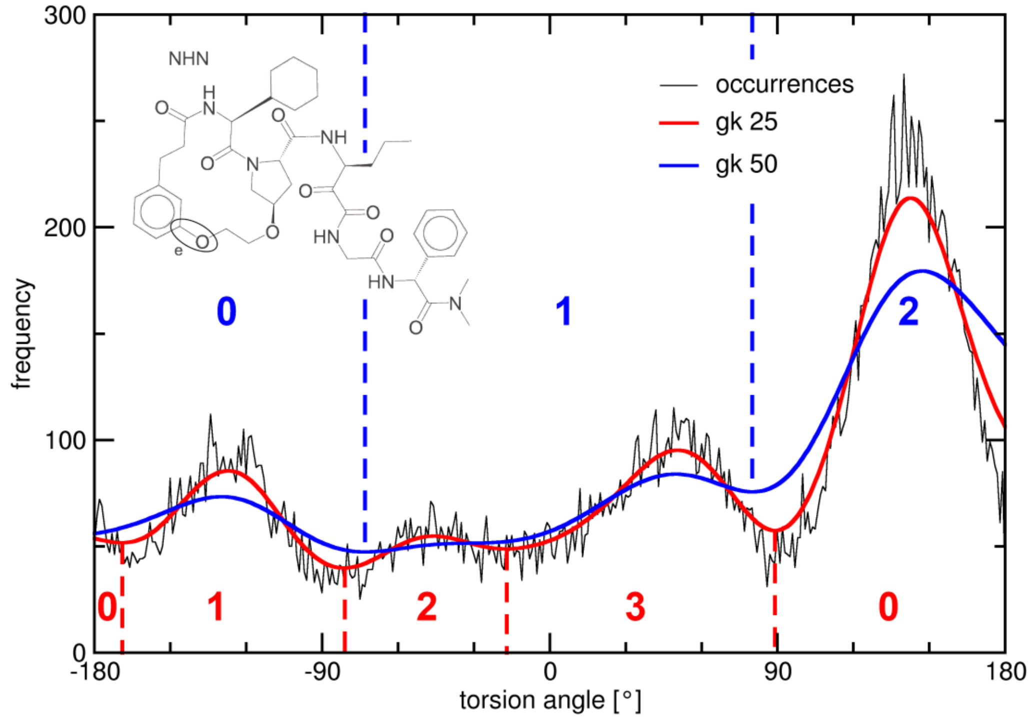


Figure 1. Torsion angle spectrum of torsion *e* of molecule NHN. Counted occurrences shown in black, different settings for the width of the gaussian kernel (*-gk*) and the corresponding division into bins shown in red and blue, respectively.

Characterizing the distribution spectrum

For the second step, the previously generated distribution spectra will be characterized to retrieve crucial information about each torsion. Therefore, the class *SpectrumAnalysis* implemented in classtor_spectrum.py calculates various objects for every torsion such as the number of bins, bin definitions, bin centers (called midpoints[1] in the following) and the status of the spectrum (see below).

An important point of this section is the definition of bins. A bin represents a part of the spectrum which groups similar values and delimits them from dissimilar ones. Together, all bins should divide the total range of torsion angle values in a meaningful way. For that, the least and most prominent regions of the torsional angle range will be identified by determining local minima and maxima of the distribution spectrum, respectively, which can be easily analyzed due to its smooth shape. ClassTor

[1] Note that the center of the bin is called midpoint but is not defined as the middle of the bin (the point equidistant from its borders) but as the most frequent torsion angle value within the bin, the local maximum.

employs the *argrelextrema*[2] function of *scipy.signal*[3,4]. This function compares each point of the spectrum with a defined number of neighboring points on both sides and defines a point as a local minimum (maximum) if its value is the lowest (highest) among its neighbors. The number of neighboring points is specified by the user *via* ClassTor's *-t* flag saved as the *Threshold* variable. In theory, the minima are used as the borders, the maxima as the midpoints of the bins.

Bin borders (minima)

For the definition of bin borders, only the local minima are used. However, since the distribution spectrum only shows the smallest window of values (from -180° to +180°) for a periodic torsion around a rotatable bond in a molecule, the "ends" of the spectrum lack neighboring points on one side and, thus, are neglected and can never be detected as extrema. Consequently, bins at spectrum borders will always miss a bin border and only bins "inside" the spectrum would be properly defined. To overcome this artifact, ClassTor automatically adds the values of the spectrum borders (-180° and +180°) to the list of local minima. Like this, the ranges of all bins can be properly defined (bin definition). The bins are labeled with integers starting from 0 beginning at -180° as exemplary shown for torsion *d* of 06H in **Table 1** and in **Figure 2** (red). From this, the number of bins is automatically detected for every torsion as the number of local (ClassTor) minima – 1, since every bin is delimited by two minima.

Table 1. Detection of local minima by *argrelextrema* and their modification with ClassTor for a general case. Correspondingly, the number of bins and their ranges as bin definition can be derived.

case	*argrelextrema*	ClassTor	bins	bin definition: [from, to]
general	[min1, min2]	[-180, min1, min2, 180]	ClassTor minima - 1	bin 0: [-180, min1] bin 1: [min1, min2] bin 2: [min2, 180]
06H-*d*	[13]	[-180, 13, 180]	2	bin 0: [-180, 13] bin 1: [13, 180]
06H-*e*	[-153, 8, 129]	[-180, -153, 8, 129, 180]	3	bin 0: [[-180, -153], [129, 180]] bin 1: [-153, 8] bin 2: [8, 129]

[2] https://docs.scipy.org/doc/scipy-1.2.3/reference/generated/scipy.signal.argrelextrema.html#scipy.signal.argrelextrema (April 12th, 2022)

[3] Jones, E., Oliphant, T., & Peterson, P. (2001). SciPy: Open source scientific tools for Python.

[4] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., ... & Van Mulbregt, P. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature methods*, *17*(3), 261-272.

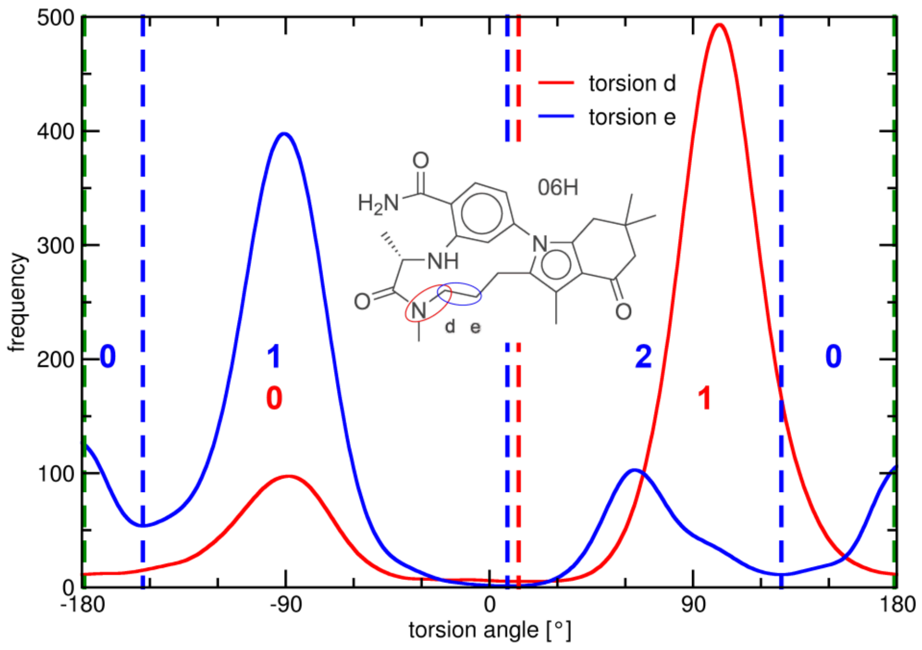


Figure 2. Visualized bin definition for the distribution spectra of torsions *d* (red) and *e* (blue) of 06H. Colored dashed lines represent bin borders (green dashed line: common bin border of both distributions) and integers bin labels of the respective distribution.

However, the artificial definition of minima at the spectrum borders represents a meaningful division only if the course shows actual minima behavior on both sides. For courses that tend towards maxima at the spectrum borders (torsion *e* of 06H, blue in **Figure 2**), the similar torsion angle values close to ±180°, described by the "broken" frequency peak around the spectrum borders, would be falsely separated into different bins. These cases need to be distinguished. Therefore, the status of a spectrum is introduced, used to adjust the bin, if needed. In equivalence to the detection of real local extrema, the *StatusInfo* function in *SpectrumAnalysis* examines both ends of the spectrum by simply comparing the outermost value (val[0] and val[n] for the left and right side, respectively) of the graph with its neighboring point (accordingly, val[1] and val[n-1]) and results in one of four options: *True*, *False*, *None* or *Limit*. Correspondingly, status of the course is defined as *(open, break)*, *(closed, clear)*, *(open, shift)* or *(closed, limit)*, respectively. **Table 2** shows in which cases the function returns each option. For very broad courses, the ends of a spectrum can be very flat and points at the borders exhibit the exact same value. Then, this distinction is unclear since only two neighboring points for each spectrum border are considered (the course of the distribution spectrum can be reshaped by altering the *GaussKern* parameter *via -gk*, see above). In this case the function compares the value at the spectrum border with every neighboring point until the first point with a different value is detected and examines cases 2-5 of **Table 2** for the new points.

Table 2. Possible cases for the tendency of the course at the spectrum borders and the corresponding return value of the function inside *StatusInfo*. The explanation is given in words for clarification and as

condition statements as implemented in ClassTor, where val[0] and val[n] are the values at the left and right border (at -180° and +180°), respectively, val[1] and val[n-1] the value of their corresponding neighboring point and *Threshold* a defined integer number specified by the user *via* ClassTor's *-t* option. The values of the first and last local maximum, which have been detected before, are defined by maxima[0] and maxima[n], respectively.

case	course behavior at borders		return
	condition	in words	value
1	val[0] = 0 **and** val[n] = 0	no values at both ends	*False*
2	val[0] < val[1] **and** val[n] < val[n-1]	course tends towards minima	*False*
3	same as 2 **and** maxima[0] <= (-180 + *Threshold*) **and** maxima[n] >= (180 - *Threshold*)	same as 2 and: maxima detected close to spectrum borders (within a range of *Threshold* points from each border)	*Limit*
4	val[0] > val[1] **and** val[n] > val[n-1]	course tends towards maxima	*True*
5	val[0] > val[1] **and** val[n] < val[n-1] **or** val[0] < val[1] **and** val[n] > val[n-1]	course tends towards minimum at one and maximum at opposite border	*None*

The status is a tuple of strings which characterizes the spectrum borders. The first value can either be "open" or "closed". In general, closed courses tend towards local minima around the spectrum borders (i.e., the values of the outermost points are smaller than the values of their neighboring points, cases 1-3, e.g. red in **Figure 2**), while open courses describe maxima (i.e., the values of the outermost points are greater than the values of their neighboring points, case 4, blue in **Figure 2**) or mixed (case 5) behavior for both borders. This information is used to remove the artificial break of torsion angle values around the spectrum borders for open courses by simply setting the bin label of the last bin at +180° to 0, thereby merging it with the first bin at -180° and, thus, virtually removing the bin borders at the ends of the spectrum (see blue bin labels in **Figure 2** and case 06H-*d* in **Table 1**). This correction provides an easy solution for a reasonable division of this types of spectrum without shifting torsion angle values *per se*. In this case, a respective remark is added to the corresponding output file (see *information* column in **Figure 5.** at the end of this section).

With the second value of the status variable, every open spectrum can be further distinguished regarding the maximum of the broken peak, which can either be located exactly at the spectrum border at ±180° (case 4, (*open*, *break*), blue in **Figure 3**) or shifted to either side of the spectrum (case 5, (*open*, *shift*), red in **Figure 3**). However, most of the time the peak will not break exactly at ±180°, but is either slightly shifted to one side, or is obtained as two separate peaks close to the spectrum borders. In the latter case (case 3), the function will detect minimum behavior at the borders, which would define a closed spectrum. Depending on the location of their maxima, the corresponding torsion angle values of those two peaks can still be considered similar. To account for this, a range of points from both borders – defined by the same *Threshold* variable (-*t*) as used for identification of local extrema – is examined and the status is set to (*closed*, *limit*), if two maxima are located within that range. Then, the first and last bins will be merged similarly as described for open courses (orange in **Figure 3**). The status of closed courses without any maxima in that range (case 1 and 2) is set to (*closed*, *clear*) and the bins can be defined without merging (green in **Figure 3**).

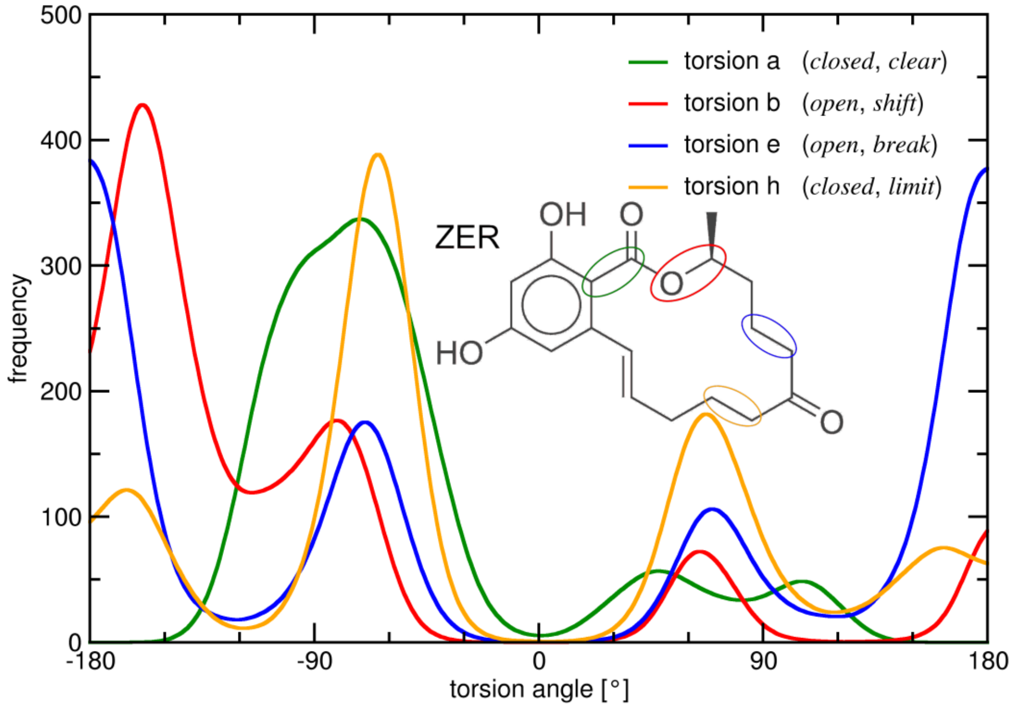


Figure 3. Distribution spectrum of four torsions of ZER (*a*: green; *b*: red; *e*: blue; *h*: orange) and their corresponding status given in brackets assigned by the *StatusInfo* function of ClassTor (-*gk* 15 -*t* 40). According to the status, first and last bins are merged for (*open*, *shift*), (*open*, *break*) and (*closed*, *limit*). The bins of a (*closed*, *clear*) spectrum can be defined without merging bins around the spectrum border.

A summary of the settings defined by the user to impact the bin definition is given in the **Supplemental Information Text 1** of the original publication.

Bin midpoints (maxima)

After the bin borders are correctly defined, midpoints are assigned to each bin. As described above, instead of taking the equidistant points between bin borders, ideally, the positions of maxima originally detected by *argrelextrema* in the previous step define more meaningful values, as they represent the most sampled torsion angle value in that part of the spectrum. However, the assignment of maxima is not trivial, e.g. for *(closed, limit)* spectra (case 3 in **Table *2***), where two bins with individual maxima were merged. To assure the correct assignment of midpoints and to make sure every bin has just one maximum, the following chain of internal functions was implemented in ClassTor:

1. *FitMax*: every local maximum is assigned to a bin according to their bin definitions.

2. *CountFittedMaxima*: for each bin, count how many maxima were assigned in step 1.

3. *AdjustMidpoints*: for torsions with status *limit* or *break,* edit maximum of bin 0.

4. *CorrectMax*: define final midpoints.

The single steps are explained in more detail as follows:

FitMax takes all previously detected local maxima and identifies which bin they belong to by comparing their position with the individual bin definitions. When the position of the maximum lies within the range of both borders of a bin, that bin label is assigned to the current maximum.

CountFittedMaxima counts how many maxima every bin got assigned. The function returns a list called *counted_maxima*, which consists of a list of integers, e.g. [2, 1, 1], where the index represents the bin and the integer the number of maxima assigned. In that example, two maxima were assigned to bin 0 and one maximum was assigned to bin 1 and bin 2, respectively. Based on this list, bins with more than one maximum can be easily identified.

Independent of the number of maxima in other bins, *AdjustMidpoints* adjusts the maxima of bin 0 for torsions with status *limit* and *break*. In both cases, bin 0 consists of two ranges that wrap around the spectrum borders, since the corresponding torsion angle values were combined into a single bin. Due to their status, we know that *limit* courses contain two maxima in bin 0. This function compares the frequency value of both and only keeps the more distinct value (the "higher" maximum). Bin 0 of a course with status *break* has no maximum assigned, since the peak breaks exactly at ±180° and was therefore not detected by *argrelextrema*. In that case, the maximum is added for bin 0 at 180°. After setting the maximum of bin 0 in both cases, the last function of the pipeline is called.

CorrectMax examines if the sum of all processed maxima is equal to the total number of bins. That is, when every bin has exactly one maximum assigned after the first 3 steps. In that case, the positions of those maxima are defined as midpoints. We implemented two more exceptions that are internally handled by ClassTor, for which a corresponding remark is added to the summary output file: either exactly one bin (that is not bin 0) has exactly two maxima assigned after the first step, or exactly one bin (not bin 0) misses one maximum. Depending on the course of the spectrum, these cases occur mostly due to inappropriate values for the *Threshold* variable (*-t*) that *argrelextrema* uses as a range of neighboring points for detection of local extrema. In the case of two maxima in one bin, a similar comparison as for *limit* spectra in step 3 is applied, keeping the maximum with the larger frequency value. In the case of no maximum in one bin, a midpoint is manually calculated as the equidistant point from both bin borders. If more than one bin has either zero or two maxima, instead of manually adjusting those midpoints, the program stops with an error message. Most likely, the course of the spectrum is too distorted, preventing the useful characterization that we aim for. Thus, the user is advised to carefully review the generated distribution spectrum of this torsion and alter the shape of the spectrum accordingly.

After the final midpoints were defined in that way, statistical analysis is performed for the following flexibility analysis. Therefore, the standard deviation of the position of all midpoints as well as the standard deviation of their frequency values are calculated with *numpy*[5].

All information about the spectrum characterization retrieved so far is saved in the corresponding output file *summary_spectrum_gk(-gk)_t(-t)_(-f).log* (with user-defined *GaussKern* (*-gk*), *Threshold* (*-t*) values and – if specified – flexible torsions (*-f*), respectively, see **Figure 5.** at the end of this section).

[5] Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357-362.

Flexibility scoring of all torsions

After all distribution spectra were characterized, ClassTor performs a flexibility assessment ranking all torsions according to their general flexibility. In case only a subset of torsions should be considered for subsequent classification, choosing the most flexible from this ranking is recommended.

For the ranking, two instances are considered: first, the overall number of bins per torsion and, second, a score calculated from the statistical information provided by the standard deviations of midpoints. According to the first instance, ClassTor grants higher flexibility to torsions with a higher number of bins compared to torsions with less bins. This is based on the assumption that a high number of bins corresponds to many distinctly separated parts of the spectrum, representing a higher potential to obtain many different dihedral angle values. Correspondingly, torsions with a single bin are not considered for ranking since – per ClassTor definition – these torsions are the least flexible.

The second instance ranks torsions with the same bin count using a score based on the statistical information retrieved from midpoints of their bins. The "FlexScore" is calculated according to **Equation 3**, which consists of a *range_score* derived from the standard deviation of midpoint positions and a *pop_score* derived from standard deviation of midpoint frequencies. The larger the FlexScore, the more flexible is the respective torsion.

$$FlexScore = range\_score * \left(pop\_score + \frac{1}{1 + pop}\right) \tag{3}$$

In general, the standard deviation of midpoint positions provides information about how the midpoints are distributed over the total range of the spectrum [-180°, +180°]. High values imply well separated midpoints that cover a broad area of the total spectrum and therefore correspond to high flexibility. In ClassTor, this is expressed by the *range_score*, which is a simple integer value ranging from 1 to the total number of torsions. All torsions with similar bin count are sorted by their value of standard deviation of midpoint positions and those integer values are assigned accordingly from lowest to highest standard deviation.

The standard deviation of midpoint frequencies (i.e., the "heights" of the peaks/maxima in the spectrum) contains information about the population of bins. Low values suggest that the heights of all bin midpoints are close to each other, corresponding to evenly populated bins, thus expressing similar occurrence of different torsion angle values, which is associated with higher flexibility. High values, however, correspond to a distinct separation of midpoint heights, implying that one or more bins and their corresponding torsion angle values are dominating the spectrum and are clearly favored over other low populated bins, which is the case for low flexibility. Therefore, ClassTor's *pop_score*, composed by an integer value ranging from 1 to the total number of torsions, is assigned to the list of torsions that is sorted from highest to lowest value of standard deviation of midpoint frequencies.

To avoid similar FlexScores among different torsions, e.g., in case the simple multiplication of their *range_score* and *pop_score* integers would result in the same value, a small fraction of the actual value of standard deviation of midpoint frequencies – *pop* in **Equation 3** – is added to the *pop_score* integer before multiplication. This fraction is larger for lower values of *pop*, thus slightly favoring more flexible torsions. This way, small nuances in FlexScores are obtained and torsions with similar levels of flexibility can be further distinguished.

Summarizing, a torsion is ranked high in flexibility by ClassTor's FlexScore when the positions of midpoints are evenly distributed over the whole range of torsion angle values (covering a broad range of values, *range_score*), and if the defined bins are equally populated (equal heights of bin midpoints, *pop_score*). The following example should clarify the FlexScore concept. All 10 torsions of macrocycle RBT, for which 3 bins were obtained by ClassTor, are assigned their *range_score* and *pop_score* integers between 1-10, respectively, according to (lowest to highest) standard deviations of midpoint positions and (highest to lowest) standard deviations of midpoint frequencies. The two most and least flexible torsions of this example are color-coded, and their spectra are shown in **Figure 4**, highlighting the differences between high and low flexibility according to ClassTor.

Table 3. FlexScores for 10 torsions of macrocycle RBT (see **Figure 4**) divided into 3 bins each. Their standard deviation values obtained from the previous characterization of distribution spectra and their corresponding *range_score* and *pop_score* integers are listed. Final FlexScores are calculated according to **Equation 3**. The two most (*c* and *e*) and least (*f* and *g*) flexible torsions are highlighted.

RBT torsion	std. dev. midpoint positions	*range_score*	std. dev. midpoint frequencies (*pop*)	*pop_score*	FlexScore
a	105.46	9	232.40	6	54.04
c	110.20	10	84.94	10	100.12
e	99.70	8	89.45	9	72.09
f	82.85	3	361.66	2	6.01
g	87.57	5	372.64	1	5.01
h	88.28	4	329.60	3	12.01
i	98.89	7	294.33	4	28.02
j	80.76	2	282.99	5	10.01
k	93.13	6	215.10	7	42.03

l	74,77	1	104.29	8	8.01

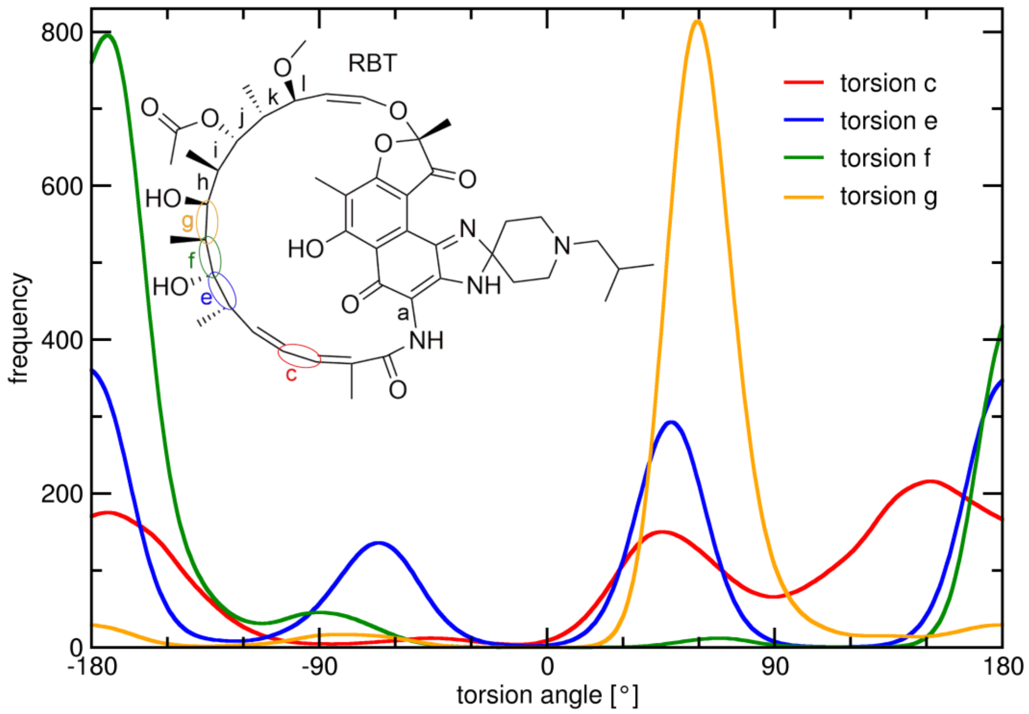
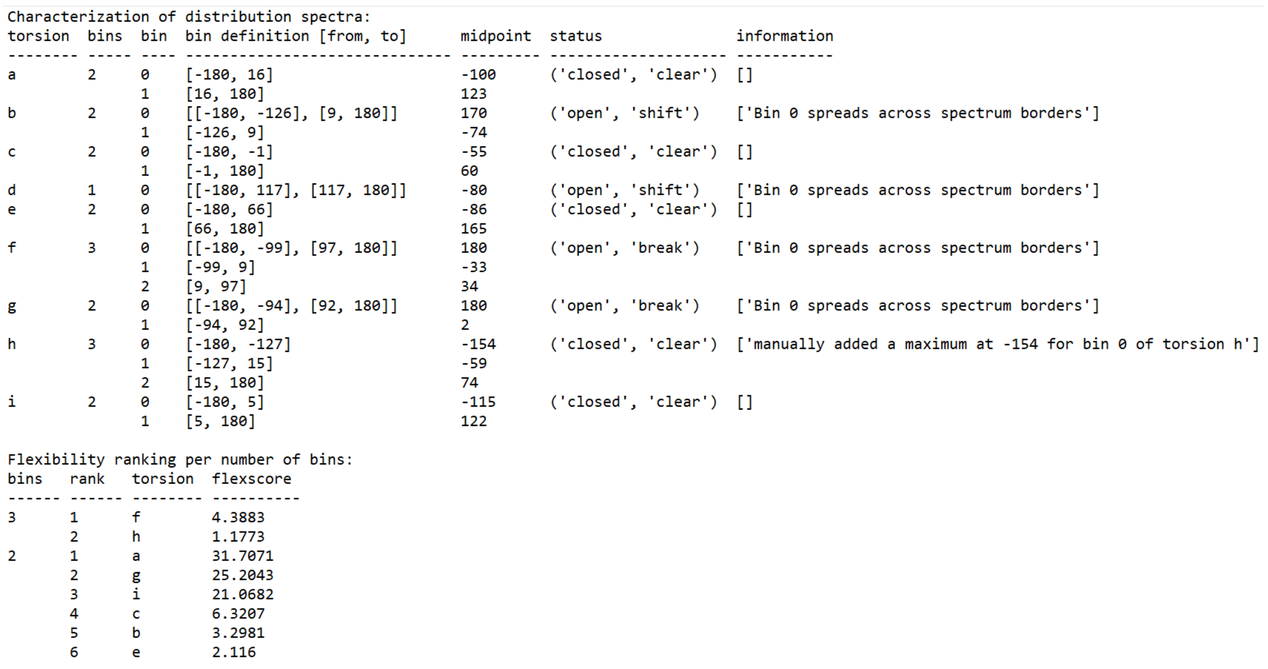


Figure 4. Distribution spectra of four torsions (*c*, red; *e*, blue; *f*, green; *g*, orange) of macrocycle RBT. Exemplary, the two most and least flexible torsions out of 10 torsions obtaining 3 bins were chosen (see **Table 3**). Illustrating ClassTor's FlexScore concept, torsions with equally populated bins and evenly distributed midpoints (*c* and *e*) are scored higher in flexibility than torsions with midpoints closer to each other, which additionally favor a single bin that dominates their spectrum (*f* and *g*).

The final ranking is added to the summary output file of the spectrum characterization, listing each torsion and its corresponding FlexScore, as exemplary shown in **Figure 5.**. After successfully generating and characterizing the torsional distribution spectrum for every torsion as well as ranking all torsions according to their flexibility, the actual classification procedure is performed in subsequent steps.



```
Characterization of distribution spectra:
torsion  bins  bin  bin definition [from, to]       midpoint  status               information
-------- ----- ---- ------------------------------ --------- -------------------- -----------
a        2     0    [-180, 16]                      -100      ('closed', 'clear')  []
               1    [16, 180]                       123
b        2     0    [[-180, -126], [9, 180]]        170       ('open', 'shift')    ['Bin 0 spreads across spectrum borders']
               1    [-126, 9]                       -74
c        2     0    [-180, -1]                      -55       ('closed', 'clear')  []
               1    [-1, 180]                       60
d        1     0    [[-180, 117], [117, 180]]       -80       ('open', 'shift')    ['Bin 0 spreads across spectrum borders']
e        2     0    [-180, 66]                      -86       ('closed', 'clear')  []
               1    [66, 180]                       165
f        3     0    [[-180, -99], [97, 180]]        180       ('open', 'break')    ['Bin 0 spreads across spectrum borders']
               1    [-99, 9]                        -33
               2    [9, 97]                         34
g        2     0    [[-180, -94], [92, 180]]        180       ('open', 'break')    ['Bin 0 spreads across spectrum borders']
               1    [-94, 92]                       2
h        3     0    [-180, -127]                    -154      ('closed', 'clear')  ['manually added a maximum at -154 for bin 0 of torsion h']
               1    [-127, 15]                      -59
               2    [15, 180]                       74
i        2     0    [-180, 5]                       -115      ('closed', 'clear')  []
               1    [5, 180]                        122

Flexibility ranking per number of bins:
bins   rank   torsion  flexscore
------ ------ -------- ----------
3      1      f        4.3883
       2      h        1.1773
2      1      a        31.7071
       2      g        25.2043
       3      i        21.0682
       4      c        6.3207
       5      b        3.2981
       6      e        2.116
```

Figure 5. Example output file *summary_spectrum_gk30_t30.log* for spectrum characterization (top) and flexibility analysis (bottom) including 9 torsions generated with ClassTor settings -*gk* 30 -*t* 30.

Torsional classification with ClassTor (classtor_classification.py)

The torsional classification implemented in the classtor_classification.py module of ClassTor consists of three main parts: the class *DoClassification,* which performs the actual classification; the function *silhouette_score*, that calculates the mean silhouette coefficient providing an internal quality measure; and the class *DiverseSubset*, which performs the so called perturbational selection extracting a diverse subset of class centroids after classification. Optionally, all class centroids can be hierarchically clustered with the *DoClustering* class to create a subset of structures. All steps will be explained in detail in the following.

Performing the classification: classes, population, centroids

DoClassification generates all classes and saves their member frames, populations, and centroids. By default, ClassTor considers all torsions specified *via* the required *AngNum* argument (see keywords section in **Usage of ClassTor.py**) for classification, unless a subset of torsions is specified *via* the *-f* option.

Using the previously determined bin definitions, the first step of the classification converts the actual torsion angle values of each torsion to the corresponding bin label of the matching bin. For example, in frame 0, the torsion angle value of torsion *a* is -48.4° and the corresponding bin definition of *a* is 0: -180° - -77°, 1: -77° - 125°, 2: 125° - 180°. The value fits between bin borders of bin 1 and is thus replaced by the label "1". This comparison is repeated for every torsion angle value of the structure and their corresponding bin definition and results in a string of bin labels for frame 0, e.g. [1, 1, 2, …], called identifier. This list encodes that torsion angle values of torsion *a* and *b* match bin 1, the value of torsion *c* fits bin 2 of their corresponding bin definition and so on. The function creates an identifier for each frame.

This translation enables a simple classification by collecting frames with the same identifiers into groups (classes). In line with the previous bin definition, where two torsion angle values of the same bin are assumed to be similar, structures with the same identifier are more similar to each other than to structures with different identifiers. Thus, the number of classes is automatically determined as the number of unique identifiers. Each class is characterized by the common identifier of their member frames, called classifier. The population of a class is simply the number of frames with the same identifier.

In a following step, the centroid of each class is determined. In ClassTor, the centroid of a class is defined as the frame with the lowest root-mean square (rms) distance between all torsion angle values and the corresponding midpoints of the bins given in the classifier. To avoid artifacts due to periodicity, every torsion angle value (and midpoint, respectively) is replaced by a sine-cosine pair after normalization. Each pair is equivalent to coordinates matching a single point on the unit circle and builds a unit vector with the origin (red and blue **Figure 6**). The difference between two torsion angle values simply corresponds to the Euclidean distance between their unit vectors, thus the norm of the distance vector (green in **Figure 6**), which is calculated with *numpy*. This way, the smallest distance between two torsion angle values is assured.

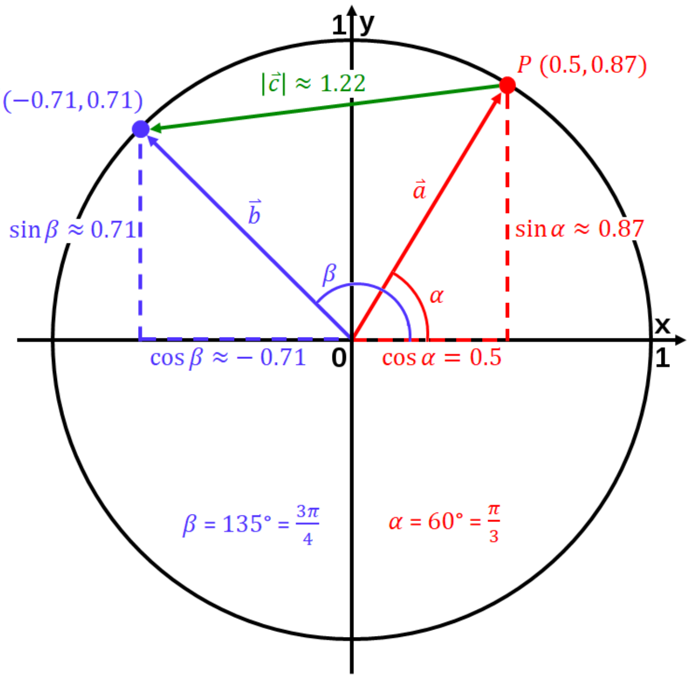
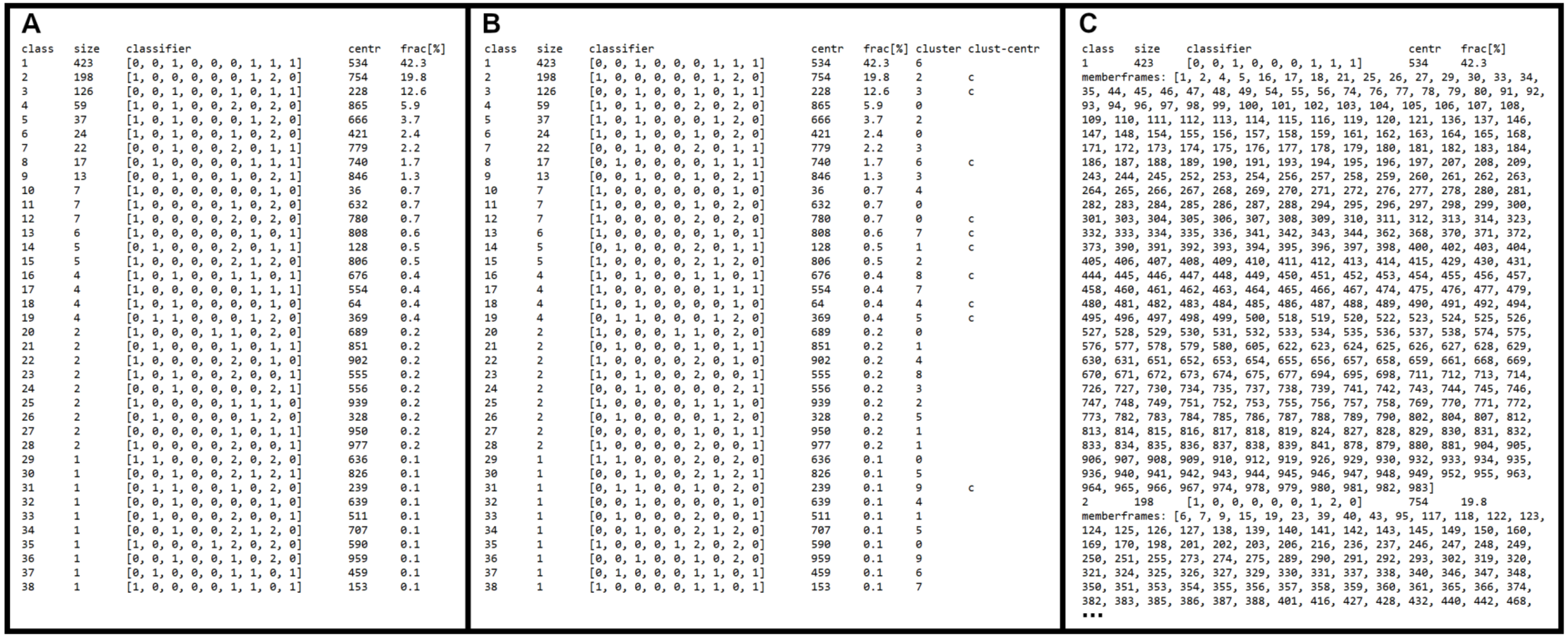


Figure 6. Relation between (torsion) angle values and coordinates of the unit circle. Example of two torsion angle values (60°, α, red and 135°, β, blue) represented as unit vectors $\vec{a}$ and $\vec{b}$ between the origin and points P and P', respectively, matching the unit circle. Coordinates of P and P' expressed as sine-cosine pairs of torsion angles α and β, respectively, after normalization. Distance vector $\vec{c}$ shown in green. Norm of the distance vector corresponds to the Euclidean distance between points P and P' and, thus, the torsion angles 60° and 135°.

After classification, the corresponding output file *summary_classification.log* is written (exemplary shown in **Figure 7**) containing the class number, size, classifier and centroid of each class as well as its fraction of the total amount of frames in percent sorted by population. Additionally, if the -*m* option is specified, the member frames of each class are added to the following line.



A

```
class   size    classifier                          centr   frac[%]
1       423     [0, 0, 1, 0, 0, 0, 1, 1, 1]         534     42.3
2       198     [1, 0, 0, 0, 0, 0, 1, 2, 0]         754     19.8
3       126     [0, 0, 1, 0, 0, 1, 0, 1, 1]         228     12.6
4       59      [1, 0, 1, 0, 0, 2, 0, 2, 0]         865     5.9
5       37      [1, 0, 1, 0, 0, 0, 1, 2, 0]         666     3.7
6       24      [1, 0, 1, 0, 0, 1, 0, 2, 0]         421     2.4
7       22      [0, 0, 1, 0, 0, 2, 0, 1, 1]         779     2.2
8       17      [0, 1, 0, 0, 0, 0, 1, 1, 1]         740     1.7
9       13      [0, 0, 1, 0, 0, 1, 0, 2, 1]         846     1.3
10      7       [1, 0, 0, 0, 0, 0, 0, 1, 0]         36      0.7
11      7       [1, 0, 0, 0, 0, 1, 0, 2, 0]         632     0.7
12      7       [1, 0, 0, 0, 0, 2, 0, 2, 0]         780     0.7
13      6       [1, 0, 0, 0, 0, 0, 1, 0, 1]         808     0.6
14      5       [0, 1, 0, 0, 0, 2, 0, 1, 1]         128     0.5
15      5       [1, 0, 0, 0, 0, 2, 1, 2, 0]         806     0.5
16      4       [1, 0, 1, 0, 0, 1, 1, 0, 1]         676     0.4
17      4       [1, 0, 0, 0, 0, 0, 1, 1, 1]         554     0.4
18      4       [1, 0, 1, 0, 0, 0, 0, 1, 0]         64      0.4
19      4       [0, 1, 1, 0, 0, 0, 1, 2, 0]         369     0.4
20      2       [1, 0, 0, 0, 1, 1, 0, 2, 0]         689     0.2
21      2       [0, 1, 0, 0, 0, 1, 0, 1, 1]         851     0.2
22      2       [1, 0, 0, 0, 0, 2, 0, 1, 0]         902     0.2
23      2       [1, 0, 1, 0, 0, 2, 0, 0, 1]         555     0.2
24      2       [0, 0, 1, 0, 0, 0, 0, 2, 1]         556     0.2
25      2       [1, 0, 0, 0, 0, 1, 1, 1, 0]         939     0.2
26      2       [0, 1, 0, 0, 0, 0, 1, 2, 0]         328     0.2
27      2       [0, 0, 0, 0, 0, 1, 0, 1, 1]         950     0.2
28      2       [1, 0, 0, 0, 0, 2, 0, 0, 1]         977     0.2
29      1       [1, 1, 0, 0, 0, 2, 0, 2, 0]         636     0.1
30      1       [0, 0, 1, 0, 0, 2, 1, 2, 1]         826     0.1
31      1       [0, 1, 1, 0, 0, 1, 0, 2, 0]         239     0.1
32      1       [0, 0, 1, 0, 0, 0, 0, 1, 0]         639     0.1
33      1       [0, 1, 0, 0, 0, 2, 0, 0, 1]         511     0.1
34      1       [0, 0, 1, 0, 0, 2, 1, 2, 0]         707     0.1
35      1       [1, 0, 0, 0, 1, 2, 0, 2, 0]         590     0.1
36      1       [0, 0, 1, 0, 0, 1, 0, 2, 0]         959     0.1
37      1       [0, 1, 0, 0, 0, 1, 1, 0, 1]         459     0.1
38      1       [1, 0, 0, 0, 0, 1, 1, 0, 1]         153     0.1
```

B

```
class   size    classifier                          centr   frac[%] cluster clust-centr
1       423     [0, 0, 1, 0, 0, 0, 1, 1, 1]         534     42.3    6
2       198     [1, 0, 0, 0, 0, 0, 1, 2, 0]         754     19.8    2       c
3       126     [0, 0, 1, 0, 0, 1, 0, 1, 1]         228     12.6    3       c
4       59      [1, 0, 1, 0, 0, 2, 0, 2, 0]         865     5.9     0
5       37      [1, 0, 1, 0, 0, 0, 1, 2, 0]         666     3.7     2
6       24      [1, 0, 1, 0, 0, 1, 0, 2, 0]         421     2.4     0
7       22      [0, 0, 1, 0, 0, 2, 0, 1, 1]         779     2.2     3
8       17      [0, 1, 0, 0, 0, 0, 1, 1, 1]         740     1.7     6       c
9       13      [0, 0, 1, 0, 0, 1, 0, 2, 1]         846     1.3     3
10      7       [1, 0, 0, 0, 0, 0, 0, 1, 0]         36      0.7     4
11      7       [1, 0, 0, 0, 0, 1, 0, 2, 0]         632     0.7     0
12      7       [1, 0, 0, 0, 0, 2, 0, 2, 0]         780     0.7     0       c
13      6       [1, 0, 0, 0, 0, 0, 1, 0, 1]         808     0.6     7       c
14      5       [0, 1, 0, 0, 0, 2, 0, 1, 1]         128     0.5     1       c
15      5       [1, 0, 0, 0, 0, 2, 1, 2, 0]         806     0.5     2
16      4       [1, 0, 1, 0, 0, 1, 1, 0, 1]         676     0.4     8       c
17      4       [1, 0, 0, 0, 0, 0, 1, 1, 1]         554     0.4     7
18      4       [1, 0, 1, 0, 0, 0, 0, 1, 0]         64      0.4     4       c
19      4       [0, 1, 1, 0, 0, 0, 1, 2, 0]         369     0.4     5       c
20      2       [1, 0, 0, 0, 1, 1, 0, 2, 0]         689     0.2     0
21      2       [0, 1, 0, 0, 0, 1, 0, 1, 1]         851     0.2     1
22      2       [1, 0, 0, 0, 0, 2, 0, 1, 0]         902     0.2     4
23      2       [1, 0, 1, 0, 0, 2, 0, 0, 1]         555     0.2     8
24      2       [0, 0, 1, 0, 0, 0, 0, 2, 1]         556     0.2     3
25      2       [1, 0, 0, 0, 0, 1, 1, 1, 0]         939     0.2     2
26      2       [0, 1, 0, 0, 0, 0, 1, 2, 0]         328     0.2     5
27      2       [0, 0, 0, 0, 0, 1, 0, 1, 1]         950     0.2     1
28      2       [1, 0, 0, 0, 0, 2, 0, 0, 1]         977     0.2     1
29      1       [1, 1, 0, 0, 0, 2, 0, 2, 0]         636     0.1     0
30      1       [0, 0, 1, 0, 0, 2, 1, 2, 1]         826     0.1     5
31      1       [0, 1, 1, 0, 0, 1, 0, 2, 0]         239     0.1     9       c
32      1       [0, 0, 1, 0, 0, 0, 0, 1, 0]         639     0.1     4
33      1       [0, 1, 0, 0, 0, 2, 0, 0, 1]         511     0.1     1
34      1       [0, 0, 1, 0, 0, 2, 1, 2, 0]         707     0.1     5
35      1       [1, 0, 0, 0, 1, 2, 0, 2, 0]         590     0.1     0
36      1       [0, 0, 1, 0, 0, 1, 0, 2, 0]         959     0.1     9
37      1       [0, 1, 0, 0, 0, 1, 1, 0, 1]         459     0.1     6
38      1       [1, 0, 0, 0, 0, 1, 1, 0, 1]         153     0.1     7
```

C

```
class   size    classifier                          centr   frac[%]
1       423     [0, 0, 1, 0, 0, 0, 1, 1, 1]         534     42.3
memberframes: [1, 2, 4, 5, 16, 17, 18, 21, 25, 26, 27, 29, 30, 33, 34,
35, 44, 45, 46, 47, 48, 49, 54, 55, 56, 74, 76, 77, 78, 79, 80, 91, 92,
93, 94, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108,
109, 110, 111, 112, 113, 114, 115, 116, 119, 120, 121, 136, 137, 146,
147, 148, 154, 155, 156, 157, 158, 159, 161, 162, 163, 164, 165, 168,
171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184,
186, 187, 188, 189, 190, 191, 193, 194, 195, 196, 197, 207, 208, 209,
243, 244, 245, 252, 253, 254, 256, 257, 258, 259, 260, 261, 262, 263,
264, 265, 266, 267, 268, 269, 270, 271, 272, 276, 277, 278, 280, 281,
282, 283, 284, 285, 286, 287, 288, 294, 295, 296, 297, 298, 299, 300,
301, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 323,
332, 333, 334, 335, 336, 341, 342, 343, 344, 362, 368, 370, 371, 372,
373, 390, 391, 392, 393, 394, 395, 396, 397, 398, 400, 402, 403, 404,
405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 429, 430, 431,
444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457,
458, 460, 461, 462, 463, 464, 465, 466, 467, 474, 475, 476, 477, 479,
480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 494,
495, 496, 497, 498, 499, 500, 518, 519, 520, 522, 523, 524, 525, 526,
527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 574, 575,
576, 577, 578, 579, 580, 605, 622, 623, 624, 625, 626, 627, 628, 629,
630, 631, 651, 652, 653, 654, 655, 656, 657, 658, 659, 661, 668, 669,
670, 671, 672, 673, 674, 675, 677, 694, 695, 698, 711, 712, 713, 714,
726, 727, 730, 734, 735, 737, 738, 739, 741, 742, 743, 744, 745, 746,
747, 748, 749, 751, 752, 753, 755, 756, 757, 758, 769, 770, 771, 772,
773, 782, 783, 784, 785, 786, 787, 788, 789, 790, 802, 804, 807, 812,
813, 814, 815, 816, 817, 818, 819, 824, 827, 828, 829, 830, 831, 832,
833, 834, 835, 836, 837, 838, 839, 841, 878, 879, 880, 881, 904, 905,
906, 907, 908, 909, 910, 912, 919, 926, 929, 930, 932, 933, 934, 935,
936, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949, 952, 955, 963,
964, 965, 966, 967, 974, 978, 979, 980, 981, 982, 983]
2       198     [1, 0, 0, 0, 0, 0, 1, 2, 0]         754     19.8
memberframes: [6, 7, 9, 15, 19, 23, 39, 40, 43, 95, 117, 118, 122, 123,
124, 125, 126, 127, 138, 139, 140, 141, 142, 143, 145, 149, 150, 160,
169, 170, 198, 201, 202, 203, 206, 216, 236, 237, 246, 247, 248, 249,
250, 251, 255, 273, 274, 275, 289, 290, 291, 292, 293, 302, 319, 320,
321, 324, 325, 326, 327, 329, 330, 331, 337, 338, 340, 346, 347, 348,
350, 351, 353, 354, 355, 356, 357, 358, 359, 360, 361, 365, 366, 374,
382, 383, 385, 386, 387, 388, 401, 416, 427, 428, 432, 440, 442, 468,
...
```

Figure 7. Different versions of the classification output file *summary_classification_gk30_t30.log*. **A)** default format, **B)** *-c* enabled; hierarchical clustering performed after classification, corresponding information is added to the output file (columns: *cluster*, *clust-centr*), **C)** *-m* enabled; member frames of each class listed below the respective entry shown for the beginning of the output file.

Calculating the silhouette coefficient

The silhouette coefficient is calculated with the *silhouette_score*[6] function as implemented in the *sklearn.metrics* module of the scikit-learn[7] library according to the definition by Rousseeuw[8]. The input to this function is an array of all x,y-coordinates per frame after conversion of torsion angle values. The silhouette coefficient of every sample of classes with more than one sample *s(i)* is determined as shown in **Equation 4**, where *a(i)* is the average (Euclidean) distance to all other samples of the same class and *b(i)* is the average distance to all samples of the nearest neighboring class.

$$s(i) = \begin{cases} 1 - a(i)/b(i), if\ a(i) < b(i) \\ 0, \quad if\ a(i) = b(i) \\ b(i)/a(i) - 1, if\ a(i) > b(i) \end{cases} \tag{4}$$

The definition in **Equation 4** leads to the following condition: -1 ≤ *s(i)* ≤ +1, where values close to +1 indicate that the sample matches its class well (that is when the average distance to samples of the same class *a(i)* is small, while the average distance to samples in the closest neighboring class *b(i)* is large). Correspondingly, values close to -1 demonstrate a bad match and indicate that the sample would fit better in the closest neighboring class (which is the case when the average distance to all samples of the closest class *b(i)* is smaller than the average distance to samples of the same class *a(i)*). For singleton classes of size 1, *s(i)* is set to 0.

It is important to mention that even if a subset of torsions is chosen for classification (defined via -*f*), the implementation of the silhouette score in ClassTor always considers the torsion angle values of all torsions (given by *AngNum*). Thus, only the resulting partition is different when changing the torsions considered for classification. With this implementation, the mean silhouette coefficient can be used for comparing different classification settings. Furthermore, independent of a centroid structure, the silhouette coefficient is suitable for comparing different classification methods, e.g., the dihedral clustering method of cpptraj (*clusterdihedral*), which performs a similar partitioning using equidistant bins and a user-defined number of bins per torsion without the definition of centroid structures. Such a comparison is shown in our study.

[6] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html?highlight=silhouette_score#sklearn.metrics.silhouette_score (April 12th 2022)

[7] Pedregosa, F. *et al.* Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* **12**, 2825-2830 (2011).

[8] Rousseeuw, P. J. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics* **20**, 53-65 (1987).

Selecting a diverse subset of class centroids

The classification method explained above provides a general tool for conformational analysis of molecular dynamics simulations of drug-like compounds and can be a first step for many computational applications. Usually, such downstream applications are computationally expensive and thus prevent the consideration of all simulated structures. Therefore, this classification reduces the simulated data in a meaningful way without losing information about the sampled space. A common way is to proceed with an ensemble of dominant conformations, e.g., the centroids of the highest populated classes. Depending on the number of torsions included for classification and the number of bins per torsion, two class centroids can still be similar to each other, for example when the bin labels in their classifier only differs for few torsions. Therefore, we implemented a different strategy that focuses on extracting a diverse subset of conformers from all class centroids after classification.

For this, the structural information from the classification is used to select centroids from classes with significantly different classifiers, i.e., classifiers that differ in bin labels for at least *P* torsions, called *P* perturbations. In short, ClassTor iteratively compares all classifiers with an initial reference classifier and determines the number of perturbations. Every classifier with less than a defined number of perturbations *P* will be rejected and is removed from the pool. The procedure is repeated in subsequent selection rounds, each starting with a new reference chosen from the remaining classifiers. This continues until a final subset remains, the classifiers of which show at least the defined *P* perturbations to each other.

The number *P*, for which the resulting subset of conformers is highly diverse, depends on the number of torsions and their flexibility. In general, the larger the number of *P*, the higher the torsional diversity of the remaining conformations, but the smaller the subset. For practical reasons, a certain subset size can be desired, e.g., when comparing different molecules. Thus, instead of giving the number *P* as input, the user specifies the final subset size via *-ss* in ClassTor. From this, *P* is automatically determined in the following way: first, the highest possible number for *P*, equal to the number of all torsions, is set and the subset is generated. If the size of the subset is smaller than defined by the user, *P* is reduced by one and the selection procedure is repeated from the beginning. This continues until the size of the final subset is equal to or greater than the requested size. In the latter case, the corresponding centroid structures of the remaining classifiers are compared to each other calculating the pairwise rms-distances between all torsion angle values and the centroids that are most different to all others are chosen as the final subset.

Besides the number of perturbations, this selection procedure is biased to the first reference classifier and the order of the remaining classifiers. In ClassTor, the former can be defined via *-sr*. If a positive integer value is specified, the classifier of the class with that number from the classification result is chosen (e.g. "*-sr* 7" will specify the classifier of the 7th highest populated class as the first reference classifier). Additionally, "0" and "-1" can be specified, where the former calculates the average classifier (= the average bin label for each torsion of all classifiers; ClassTor default) and the latter defines a random classifier from the classification results as the first reference. Note that by implementation, the centroid structure of the first reference classifier is always part of the resulting subset. If *-sr 0* is specified and the calculated average classifier matches with an actual existing classifier of the classification result, this classifier is chosen as the first reference. The order of all classifiers for the first selection round is specified via *-so* and the options are: "random" (ClassTor default), "topdown" and "reverse", where the order of classifiers is random, from highest to lowest and from lowest to highest class population, respectively.

Depending on the objective, it can be desired to keep certain conformations. For example, with *-sr* 1 it is guaranteed that the highest populated class centroid (the most dominant conformation) will be a member of the resulting subset. In combination with *-so topdown* this can bias the selection concentrating on the more populated classes. Correspondingly a combined input of the classifier of the last class and *-so reversed* favors less populated classes to remain in the subset. Thus, a careful analysis and visual comparison of all class centroids after classification is recommended to gain information about the results that can direct the subset extraction. In our study, the least biased selection and highest diversity was achieved with the default combination *-sr* 0 and *-so random*. With these settings the resulting subsets are not reproducible (*-so random*). However, we found that different subsets extracted in that way showed similar degree of conformational diversity, in line with a robust procedure.

ClassTor provides the centroid frames of the conformers of the resulting diverse subset as standard output together with the similar number of centroids of the highest populated classes for comparison. Additionally, the (average) reference classifier is given (if *-sr 0*) with a note if this classifier existed or if it was virtual. As a summary note, the final number of perturbations *P* is given, that created the subset of conformers.

To visualize the conformational diversity of the extracted subset, ClassTor writes two output files: a distance matrix that contains the pairwise rms-distances in torsion angle space (d-RMSD) between a list of frames (*RMS2TA_(-f)_(-ss)class_(-ss)diverse.gnu*) and the corresponding gnuplot input file plotting this matrix as a heatmap (*heatmap_(-f)_(-ss)class_(-ss)diverse.in*: see *splot* command); by default, the distance matrices of two separate conformer subsets are calculated for comparison: the centroids of the diverse subset (column 4 in *.gnu*) and the corresponding (*-ss*) number of class centroids of the highest populated classes (column 3 in *.gnu*). The heatmap file can be read by gnuplot:

```
gnuplot -p < heatmap.in
```

By default, the distance matrix of the diverse centroids is plotted (columns 1, 2 and 4 of *.gnu*, see *splot* command in *heatmap.in*: "using 1:2:4"). By simply changing this command to "using 1:2:3" the corresponding distance matrix between the centroids of the highest populated classes are plotted. If *-x* is specified, the last frame (last line in every torsion angle file) is considered as a reference and added to both list of frames for which the distance matrices are calculated. The torsion angle values of the reference structure are excluded from the previous ClassTor steps (spectrum generation, spectrum characterization and classification). For clarity, this reference frame is separated by black lines in the heatmaps. This way, an easy comparison between different subsets of conformers is feasible, thus visualizing the conformational diversity within each set. Exemplary shown for the macrocyclic molecule radicicol (RDC), **Figure 8A** and **8C** show the heatmaps of conformer subsets comprising the 10 highest populated class centroids and 10 diverse class centroids, respectively. The color-coded d-RMSD indicates highly diverse conformers (red) and similar structures (blue).

Hierarchical clustering of classification centroids

As an alternative to the above mentioned perturbational approach for the extraction of conformer subsets, the hierarchical agglomerative clustering approach of the scipy library (scipy.cluster.hierarchical[9]) was implemented within the class *DoClustering* in classtor_classification.py. Specifying the flag *-c* enables clustering, which is performed after classification considering all class centroid structures. The clustering is stopped when the desired number of clusters is reached, specified via *-ss*. Distances between samples are calculated as the Euclidean distances between their unit circle coordinates representing the converted torsion angle values (see above) and the Ward's linkage method is employed. A centroid is determined for each cluster as the structure with the minimum rms-distance to all other members of the cluster. Clustering information is added to the *summary_classification.log* output file (**Figure 7B**) specifying the cluster each class centroid belongs to (column *cluster*) and the centroid of each cluster (marked with a "c" in the column *clust-centr*). This way, the results of classification and subsequent clustering are given in the same file for easy inspection. Additionally, for the thus generated subset of cluster centroids, a similar comparison is performed

[9] https://docs.scipy.org/doc/scipy-1.2.3/reference/cluster.hierarchy.html#module-scipy.cluster.hierarchy

calculating a d-RMSD matrix writing the output file *RMS2TA_(-f)_(-ss)class_**(-ss)clus**_(-ss)diverse.gnu*, where column 3 specifies the subset of highest populated class centroids, column 4 the subset of clustered class centroids and column 5 the subset of diverse class centroids extracted with the perturbational approach. Accordingly, changing the *splot* command in the corresponding *heatmap_(-f)_(-ss)class_(-ss)clus_(-ss)diverse.in* ("using 1:2:3/4/5") controls which subset will be plotted. Exemplary, **Figure 8** shows the heatmaps of all 3 subsets for a subset size of 10 structures each. The subset of the 10 highest populated class centroids (**Figure 8A**) shows large blue areas (low torsional diversity between structures), the subset of 10 cluster centroids (**Figure 8B**) shows moderate diversity (red area). However, the subset of 10 class centroids extracted with ClassTor's perturbational approach (**Figure 8C**) is the most diverse, while still featuring a conformer close to the reference (x, here: the bound conformation, see blue spot between x and class centroid 1). Depending on the objective of your study, the corresponding subsets can be extracted. If you want to cover the most dominant structures, the set of highest populated class centroids is useful. If you are looking for the structurally most diverse conformers, ClassTor's perturbational approach is suitable.

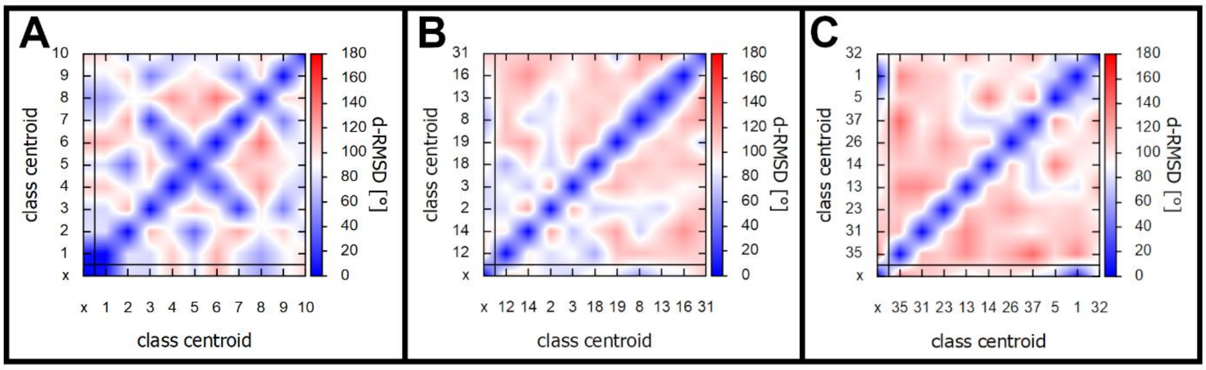


Figure 8. Pairwise rms-distance matrix in torsion angle space (d-RMSD) between a reference (x, here: bound ligand conformation) and a subset of 10 conformers (ClassTor output files: *heatmap_10class_10clus_10diverse.in*, *RMS2TA_10class_10clus_10diverse.gnu*). The plots show heatmaps with high d-RMSD values (diverse structures) in red and low values (similar structures) in blue. **A)** subset contains the first 10 class centroids, **B)** subset consists of 10 clustered class centroids, see **Figure 7B**, **C)** subset extracted with ClassTor's perturbational approach (*-sr 0* and *-so random*).

Usage of ClassTor.py

This section provides detailed instructions for the usage of ClassTor. An overview of all required and optional keywords is listed that can be used with the ClassTor.py script.

Command line keywords

```
ClassTor.py AngNum [OPTION]
```

	AngNum	required integer number of torsions; must be the first argument. The corresponding amount of individual two-column files with the dihedral angle value of a torsion per frame must be labelled `X_angles.dat`, where `X` is the letter of a labeled torsion starting with `a`. 0 < *AngNum* <= 26
preparation	*-pa STR*	path to location of torsion angle files (optional, default: `./`)
	-x	when specified, the last line of every torsion angle file is taken as a reference value and is only included for comparison calculations to that reference structure, not for the generation of each torsion angle spectrum or classification (optional, default: not specified)
spectrum	*-gk INT*	width of the gaussian kernel used for convoluting the frequency of each torsion angle value when generating the corresponding spectrum (optional, default: 15)
	-t INT	number of neighboring data points of the spectrum considered for the detection of relative extrema (minima and maxima) and the characterization of the course at the spectrum borders (at ±180°, optional, default: 20)
classification	*-f STR*	string of letters separated by whitespace specifying a subset of torsions used for classification, e.g., when only the most flexible torsion should be considered. If not specified, all *AngNum* torsions are considered (optional, default: not specified)
	-m	when specified, the class members of each class are added to the classification output file *summary_classification.log* to a separate line under the corresponding class (optional, default: not specified)
subset extraction	*-ss INT*	size of the subset of conformers extracted after classification (optional, default: 10)
	-sr INT	specifies the first reference classifier for the perturbational subset extraction: 0 < *INT* ≤ highest class-ID: the classifier of the specified class-ID is used (first reference is always included in the subset)

	0: the average classifier is determined of all classes obtained after classification and used as a first reference (default, will be included in the subset if the average classifier matches an existing classifier of the classification) -1: the classifier of a random class (0 < *INT* ≤ highest class-ID) is used
-so STR	specifies the order in which all classifiers from the classification are compared to the first reference (*-sr*) in the first selection round: “random”: shuffle the list of classifiers to randomize the order (default) “topdown”: start from highest to lowest populated class “reverse”: start from lowest to highest populated class

clustering

-c	when specified, hierarchical agglomerative clustering is performed with centroid structures after classification. This is an alternative approach of generating a subset of conformers with size *-ss* from the classification results (optional, default: not specified)