PyCPR – A Python-based Implementation of the Conjugate Peak Refinement (CPR) Algorithm for Finding Transition State Structures

Florian J. Gisdon,[*] Martin Culka,[*] G. Matthias Ullmann

[*] both authors contributed equally

Computational Biochemistry, University of Bayreuth, Universitätsstr. 30, NW I, 95447 Bayreuth, Germany

Correspondence should be addressed to: Matthias.Ullmann@uni-bayreuth.de

Manual

If you use this software for a published work, please cite:

Florian J. Gisdon, Martin Culka, G. Matthias Ullmann (2016): PyCPR – A Python-based Implementation of the Conjugate Peak Refinement (CPR) Algorithm for Finding Transition State Structures.
J. Mol. Model. **22**: *242*, 2016
DOI: 10.1007/s00894-016-3116-8

1. Incorporating PyCPR into pDynamo

The PyCPR[1] implementation of the Conjugate Peak Refinement (CPR) algorithm[2] consists of three Python modules. The modules need to be copied into the appropriate directories of the working pDynamo[3] distribution and the corresponding `__init__.py` of those folders have to be adapted. Here is the list of PyCPR modules and their destinations in the pDynamo tree:

Module file	Destination directory
ConjugatePeakRefinement.py	pMoleculeScripts
CPRSaddlePointRefinement.py	pCore
MoreThuenteLineSearchWithMax.py	pCore

The installation can be done automatically by the `install.csh` script, provided that the pDynamo environmental variables are correctly set.

Alternatively, the installation can be done manually. First, please make sure that you have a working copy of `pDynamo` installed on your system. Copy the files from the directory `modules` to appropriate `pDynamo` directories:

- `ConjugatePeakRefinement.py` to `$PDYNAMO_PMOLECULESCRIPTS/pMoleculeScripts`
- `CPRSaddlePointRefinement.py` and `MoreThuenteLineSearchWithMax.py` to `$PDYNAMO_PCORE/pCore`

In this case, the `__init__.py` files in the corresponding directories have to be modified accordingly, i.e. the following lines need to be added:

- to `$PDYNAMO_PMOLECULESCRIPTS/pMoleculeScripts/__init__.py`

```
from ConjugatePeakRefinement  import ConjugatePeakRefinementOptimizePath
```

- to `$PDYNAMO_PMOLECULESCRIPTS/pMoleculeScripts/__init__.py`

```
from CPRSaddlePointRefinement      import CPRSaddlePointRefinement
from MoreThuenteLineSearchWithMax  import MoreThuenteLineSearchWithMax
```

After a successful installation, the example script that finds a reaction path for the conformational change in butane molecule can be run. The script `butane-CPR.py` and the input structures can be found in the directory `example-butane`.

```
python butane-CPR.py > butane-CPR.out
```

2. Running Conjugate Peak Refinement

Once a chemical system and an initial trajectory (at least 2 path points) is defined within the pDynamo framework, the PyCPR path search can be run calling following function:

```
ConjugatePeakRefinementOptimizePath (system, imageTrajectory, \
                                              **keywordArguments)
```

where

- `system` is a pDynamo data structure containing the definition of the system
- `imageTrajectory` is a pDynamo data structure containing the initial reaction path
- `**keywordArguments` is a list of optional keywords for PyCPR which are explained in the following

2.1. Getting started

Alongside with the PyCPR modules, we provide a simple example script for calculation of the conformational change from anti to gauche in a butane molecule. This script can be used as a basis for setting up own calculations using the CPR algorithm. In this example, a pure quantum-mechanical (QM) system in gas phase is defined using the semiempirical RM1 method as implemented in pDynamo.

```
#!/usr/bin/python

from pBabel import MOLFile_ToSystem, XYZFiles_ToSystemGeometryTrajectory, \
                   SystemGeometryTrajectory, \
                   XYZFiles_FromSystemGeometryTrajectory
from pMolecule import QCModelMNDO, ADIISSCFConverger, DIISSCFConverger, \
                      ElectronicState
from pMoleculeScripts import ConjugatePeakRefinementOptimizePath

#------------------------------------------
# Define the molecule and its energy model
#------------------------------------------
converger=ADIISSCFConverger(densityTolerance=10e-13 )
system = MOLFile_ToSystem( "./input/butane.mol"  )
system.electronicState = ElectronicState(charge = 0, multiplicity = 1)
qc_model = QCModelMNDO("rm1", converger=converger)
system.DefineQCModel (qc_model)
system.Summary ()

#------------------------------------------------------
# Generate initial trajectory (at least 2 structures!)
#------------------------------------------------------
```

```
initial_path = (
"./input/butane-anti-opt.xyz",
"./input/butane-gauche-opt.xyz",
)

traj_path = "cpr_traj"    # directory to save the path structures to

XYZFiles_ToSystemGeometryTrajectory(initial_path, traj_path, system)
trajectory     = SystemGeometryTrajectory(traj_path, system, mode = "a+")

#-----------------------------------------------------------------
# Run CPR on the *system* starting from the initial *trajectory*
#-----------------------------------------------------------------
ConjugatePeakRefinementOptimizePath ( system, trajectory, \
                                      rmsGradientTolerance = 0.001, \
                                      breakIfTauReached = False, \
                                      interpolationIncreasement = 0, \
                                      finalUnrefinableRefinement = False)

#----------------------------------------
# Convert the final path to .xyz formate
#----------------------------------------
XYZFiles_FromSystemGeometryTrajectory ( traj_path, traj_path, system )
```

Within the pDynamo framework, it is also possible to set up different kinds of calculations, e.g. molecular-mechanical (MM) or hybrid QM/MM. This has been already described in following pDynamo documentation sources:

- book by M. J. Field, *A Practical Introduction to the Simulation of Molecular Systems* published by Cambridge University Press in 2007
- pDynamo wiki webpage
  `http://sites.google.com/site/pdynamomodeling/getting-started`

2.2. Adjustable algorithm parameters

The CPR calculation can be adjusted by changing the default algorithm parameters. Please note that although many parameters of the CPR algorithm are adjustable, the default values are usually chosen reasonably and a significant modification may cause that the resulting path has little meaning. The user is advised to carefully read the paper describing PyCPR[1] in order to understand the theory behind the parameters before changing the default values. The adjustable parameters can be categorized according to the three major domains of the algorithm (as described in Ref. 1). A fourth group gathers parameters that are related to the overall course of the CPR algorithm.

Highest Point Search. In this part, the CPR algorithm searches for the point (structure) with the overall highest energy along the reaction path to be later optimized. The path from the reactant to the product state is approximated by piecewise linear interpolations between the path points. The extend of this discretization is dependent on the length of the path segment. The minimal number of discretization steps can be adjusted by setting the *stepsPerSegment* parameter.

The adjustable parameter for the *Highest Point Search* domain of the CPR algorithm is:

Parameter	Default	Short description
stepsPerSegment	3	Determines the minimal number of discretization steps

Saddle Point Refinement. The highest point of the path is passed to the *Saddle Point Refinement* procedure each CPR cycle. The full theory description can be found in Ref.1. Here only the necessary parts are briefly repeated.

The first step of the *Saddle Point Refinement* is a line maximization of the initial point $\mathbf{x}_0$ along the initial search direction $\mathbf{s}_0$, which is a tangential direction to the path at this point. Point $\mathbf{x}_1$ representing a local maximum along $\mathbf{s}_0$ is thus obtained. As described in the theory section of Ref.1, the difference between the gradient at the point $\mathbf{x}_0$ ($\mathbf{g}_0$) and $\mathbf{x}_1$ ($\mathbf{g}_1$) is used to construct a set of conjugate minimization directions $\mathbf{s}_i$. This way of constructing the directions is used if the parameter *finiteDifference* is set to *False*. However, if the path is not yet well sampled, the difference between $\mathbf{x}_0$ and $\mathbf{x}_1$ is often significant and for the conjugacy term it is more advisable to use a finite difference approximation. Thus when the parameter *finiteDifference* is set to *True*, a step of the length specified in *finiteDifferenceStep* is taken from $\mathbf{x}_1$ along $\mathbf{s}_0$ and the gradient of this point is taken as $\mathbf{g}_0$ for the construction of conjugate directions $\mathbf{s}_i$. Another parameter for the construction of the set of conjugate optimization directions ($\mathbf{s}_i$) is the scaling factor beta-type. In our implementation, three different beta-types can be selected, please refer to Ref.1 for details.

If no nearby maximum is found along $\mathbf{s}_0$, and $\mathbf{x}_0$ was an existing path point, the point gets deleted from the path. The criterion for being nearby is evaluated based on RMSD and can be adjusted by changing the parameter *rmsdToleranceMaximum*. If a nearby maximum is found, the resulting point $\mathbf{x}_1$ is further optimized by a series of conjugate line minimizations along gradually constructed directions $\mathbf{s}_i$. The initial step of the line search can be adjusted by the parameter *lineSearchInitialStep* (affects also the initial line maximization). During the optimization process, occasionally a low gradient region is reached where the RMS gradient is below the *rmsGradientTolerance* threshold. Once the gradient stays low for M steps, a saddle point is found. By default, M is set to $\sqrt{N}$ where N is the number of atoms in the system. Alternatively, M can be directly specified by the parameter *numberOfSuccessiveLowGradients*. It might seem that the default number of the *rmsGradientTolerance* threshold is too high and the *numberOfSuccessiveLowGradients* is too big. However, to locate a saddle point, it is important to check more search directions within the low gradient region. If the *rmsGradientTolerance* would be too low, one could run into a numerical precision problem, since the gradient gets anyway significantly below this threshold if the *numberOfSuccessiveLowGradients* is big enough.

During the conjugate optimization process, it can happen that the next search direction is no longer conjugate to $\mathbf{s}_0$. After the parameter τ (see Ref.1 for details) reaches a certain tolerance t, the optimization may be interrupted. Since it is advisable to allow bigger flexibility when adding a point from the linear interpolation compared to when refining an existing path point, two thresholds are introduced – *tauTolAdd* and *tauTolRefine*, respectively. When the path is not well refined yet, it might be beneficial to allow the algorithm to find new possible minima by breaking the conjugacy criterion during the optimization. This behavior can be allowed by setting the parameter *breakIfTauReached* to *False*. Sometimes it happens, that the search directions do not lead to a low gradient region, yet the directions remain conjugate or *breakIfTauReached* is *False*. By default, the CPR algorithm optimizes along all theoretically reasonable $3N-1$ conjugate directions. The user can reduce this number by setting the parameter *maximalMinimizationSteps*, yet this number should be set reasonably big (at least two times the *numberOfSuccessiveLowGradients* in order not to break in the low gradient region before reaching the saddle point criterion).

The adjustable parameters for the *Saddle Point Refinement* domain of the CPR algorithm are:

Parameter	Default	Short description
finiteDifference	`True`	use finite difference for x_0 reevaluation
finiteDifferenceStep	`1.0e-3`	length of the finite difference step
betaType	`PRP`	specify betaType `FR`: Fletcher and Reeves `PRP`: Polak, Ribière and Polyak `HS`: Hestenes and Stiefel
rmsdToleranceMaximum	`1.5`	threshold for the nearby maximum (in Å)
lineSearchInitialStep	`0.1`	length of the first line search step (in Å)
rmsGradientTolerance	`0.1`	threshold for entering the low gradient region when approaching the saddle point (in kJ/(mol Å)
numberOfSuccessiveLowGradients	`None`	number of successive low gradients during the optimization in order to classify the optimized structure as a saddle point; `None` means that $\sqrt{N}$ taken
tauTolAdd	`0.15`	τ exit threshold when adding a new point to the path
tauTolRefine	`0.05`	τ exit threshold when refining an existing path point

breakIfTauReached	`True`	if `True`, this CPR cycle stops if τ threshold is reached; if `False` the cycle continues, but the resulting point is not marked as saddle point
maximalMinimizationSteps	`None`	Max. number of optimization steps within a CPR-cycle; `None` means that the theoretical maximum ($3N - 1$) is taken

Futile Loop Prevention. During the CPR path optimization, a futile adding and deleting of the path points might occur. In Ref.1, we describe a strategy to prevent such infinite loops within the *Futile Loop Prevention* section.

A futile loop occurs, if a ($\mathbf{x}_0$, $\mathbf{s}_0$) couple, that has already been treated, is encountered again. In this case, the optimization strategy is modified. The similarity is determined by RMSD and the threshold can be adjusted by the parameters *rmsdLoopPreventionX0* and *rmsdLoopPreventionS0*. The first prevention strategy is to rediscretize the segment with a bigger number of steps and thus hopefully find better $\mathbf{x}_0$ to be optimized. The extend of this increase can be adjusted by the parameter *interpolationIncreasement*. Doubling of the τ threshold t is another strategy and this can be switched on or off by the parameter *increaseTau*.

The adjustable parameters for the *Futile Loop Prevention* domain of the CPR algorithm are:

Parameter	Default	Short description
rmsdLoopPreventionX0	`1.0e-6`	RMSD tolerance of $\mathbf{x}_0$ for futile loop detection in Å
rmsdLoopPreventionS0	`1.0e-6`	RMSD tolerance of $\mathbf{s}_0$ for futile loop detection in kJ/(mol Å)
interpolationIncreasement	`3`	How much to increase the linear discretization of a path segment during the futile loop prevention
increaseTau	`False`	if `True`, the τ-threshold for exiting the *Saddle Point Optimization* is doubled as strategy for the futile loop prevention

General algorithm parameters. In addition to the specific settings, several general parameters may be adjusted:

- It is possible to change the maximal number of the CPR cycles by changing the option *maxCPRrun*.

- In the end of the CPR procedure, it is possible to revisit the points marked as unrefinable (*finalUnrefinableRefinement*).

- When there is no good initial guess of the path, it might be advisable to run first few cycles using orthogonal directions instead of conjugate (*initialOrthogonalRuns*).
- When an external program (e.g. ORCA) is used for the QM part and an electron transfer is happening during the studied reaction, it might be advisable to clear the scratch directory after every CPR cycle by setting the *scratchDirectory* to the path of this directory in order to avoid restarting from the previous electronic state.
- A restart file containing information of the highest point in every current path segment as well as the saddle, stationary and unrefinable point flags is written out every CPR cycle. This file is by default called `cpr.restart` and this name can be changed by the parameter *outputRestartFile*. When restarting after a previous CPR run, the restart file corresponding to the structures from the previous run may be loaded in (path specified by *inputRestartFile*).

The general adjustable CPR parameters are:

Parameter	Default	Short description
maxCPRrun	`1000`	Max. number of CPR-cycles
finalUnrefinableRefinement	`False`	If `True`, points marked as unrefinable are treated again once before exit
initialOrthogonalRuns	`0`	Number of initial CPR-cycles optimizing using orthogonal directions instead of conjugate.
scratchDirectory	`None`	If specified the folder is cleaned up after every cycle.
outputRestartFile	`cpr.restart`	Path for a restart file to be written
inputRestartFile	`None`	Path for a restart file to be read

2.3. General recommendations

It is possible to start the reaction mechanism search by PyCPR providing only an initial and a final structure as an input. If you do so, please keep in mind that CPR starts from a linear interpolation between the provided initial path points and if the structures are very different, the optimization might fail in an early stage e.g. due to electronic structure convergence difficulties. Even if this does not happen, one has to keep in mind that although the resulting path contains low gradient points, the search might be biased by the initial linear guess and the resulting mechanism is not necessarily the one with minimal barrier. In case of complex reactions it is thus more advisable to provide an initial guess for the path. If some stable intermediates are already known e.g. from a potential energy surface scan, they can help to guide the algorithm to find the saddle points between them. Alternatively, CPR can be combined with another reaction path search method, e.g. the growing string and/or nudged elastic band methods within pDynamo. The trajectory (as the second CPR parameter, *imageTrajectory*) can be passed directly between the methods. For more detailed discussion about the recommended Py-CPR usage strategy please refer to Ref.1.

2.4. Treating problematic cases

Some recommendations are already given in Ref.1. Here, we provide a few more practical hints.

During the reaction path search with PyCPR, difficult regions can occur on the energy landscape, where the algorithm may fail to proceed. This behavior could occur in a rather flat region, where still a slightly higher maximum is found within the segment after refinement of a previously found maximum. Such a behavior can lead to accumulation of very similar points. If the energetic difference of these points is vanishing, it is advisable to pick a representative point of that region, delete almost equal ones and mark the neighboring segments as resolved using the restart file. Then the algorithm can concentrate on other regions of the path and one can treat the accumulation region separately. If points marked as unrefinable still exist at the end of the CPR cycles, especially if such a point is the highest along the path, one should treat them individually. For that, it is worth to generate further images in this problematic segment to get a better sampling and refine them. This approach creates new starting points for the optimization and a changed surrounding, which mostly improves the refinement of these regions. In general, the transition state search is an interactive progress and the result has to be analyzed critically.

References

[1] Gisdon, F. J.; Culka, M.; Ullmann, G. M. *J. Mol. Model.*, **2016**, *22*, 242.

[2] Fischer, S.; Karplus, M. *Chemical Physics Letters*, **1992**, *194*, 252–261.

[3] Field, M. J. *J. Chem. Theory Comput.*, **2008**, *4*, 1151–1161.