RDC-ANALYTIC
User Manual
Version 1.0

Chittaranjan Tripathy and Bruce R. Donald

Contents

1 Introduction

RDC-ANALYTIC is a suite of programs for high-resolution protein backbone fold determination from residual dipolar couplings (RDCs) (only two RDCs per residue are required) in one alignment medium, and a sparse set of nuclear Overhauser effect (NOE) data. RDC-ANALYTIC is developed in the lab of Prof. Bruce R. Donald at Duke University.

In an earlier prototype of our software the name RDC-EXACT was used (since low-degree polynomial equations derived from RDC equations can be solved *exactly* in a mathematical sense). This later version is named RDC-ANALYTIC to emphasize the analytic (exact) solutions to the RDC equations, and to avoid the confusion that this software cannot (of course) guarantee biological "exactness".

RDC-ANALYTIC is free software and can be redistributed and/or modified under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (optionally) any later version. RDC-ANALYTIC is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. For full licensing details, including citation requirements for the software, please refer to Section 2 and Section 3, respectively. This information can also be found in the document `license.pdf` enclosed with this package distribution.

RDC-ANALYTIC is designed to compute high-resolution global fold of proteins from RDCs in one alignment medium. It combines analytic solutions to RDC equations with a systematic depth-first search based technique to determine the structures of secondary structure elements (SSEs) of a protein and their relative orientations with respect to a global reference frame. The SSEs computed by RDC-ANALYTIC can be packed using a sparse set of NOEs to output the global fold of a protein. Although RDC-ANALYTIC uses at least two RDCs (one N-H^N and one C^α-H^α RDC) per residue, we expect it to perform well even when only approximately 90% of the N-H^N and C^α-H^α RDCs are present. For the cases with less RDC data some heuristic techniques can be employed to compute the structure but the results may be less accurate since the system with less data becomes highly under-determined.

Currently, RDC-ANALYTIC is the first module in the RDC-PANDA high-resolution protein structure determination software from our lab [1]. We highly recommend the users to use RDC-PANDA[1] with which RDC-ANALYTIC comes as a built-in package. This document contains license information, citations required upon using the software, and the details of how to install and use RDC-ANALYTIC.

2 License Information

The source header below must be included in any modification or extension of the source code of RDC-ANALYTIC.

Source Header

```
This file is part of RDC-ANALYTIC.

RDC-ANALYTIC Protein Backbone Structure Determination Software Version 1.0
Copyright (C) 2001-2009 Bruce Donald Lab, Duke University
```

```
RDC-ANALYTIC is free software; you can redistribute it and/or modify it under
the terms of the GNU Lesser General Public License as published by the Free
Software Foundation, either version 3 of the License, or (at your option) any
later version.

RDC-ANALYTIC is distributed in the hope that it will be useful, but WITHOUT
ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more
details.

You should have received a copy of the GNU Lesser General Public License
along with this library; if not, see:
    <http://www.gnu.org/licenses/>.

There are additional restrictions imposed on the use and distribution of this
open-source code, including: (A) this header must be included in any
modification or extension of the code; (B) you are required to cite our
papers in any publications that use this code. The citation for the various
different modules of our software, together with a complete list of
requirements and restrictions are found in the document license.pdf enclosed
with this distribution.

Contact Info:
    Bruce R. Donald
    Duke University
    Department of Computer Science
    Levine Science Research Center (LSRC)
    Durham, NC 27708-0129
    USA
    email: www.cs.duke.edu/brd/

<signature of Bruce Donald>, 01 December, 2009
Bruce R. Donald, Professor of Computer Science and Biochemistry
```

3 Citation Requirements

Any publications, grant applications, or patents that use RDC-ANALYTIC must state that RDC-ANALYTIC was used, with a sentence such as "We used the open-source RDC-ANALYTIC software [Ref] to compute..."

In addition, you are required to cite our papers in any publications that use this code. The primary citation corresponding to this software is [1]. The papers that can be cited based-on or related-to this software are listed below.

[1] Jianyang Zeng, Jeffrey Boyles, Chittaranjan Tripathy, Lincong Wang, Anthony Yan, Pei Zhou, and Bruce Randall Donald. High-resolution protein structure determination starting with a

global fold calculated from exact solutions to the RDC equations. *Journal of Biomolecular NMR*, 45(3):265–281, 2009.

[2] Bruce R. Donald and Jeffrey Martin. Automated NMR Assignment and Protein Structure Determination using Sparse Dipolar Coupling Constraints. *Progress in Nuclear Magnetic Resonance Spectroscopy*, 55(2):101–127, 2009.

[3] Lincong Wang, Ramgopal R. Mettu, and Bruce R. Donald. A Polynomial-Time Algorithm for *De Novo* Protein Backbone Structure Determination from NMR Data. *Journal of Computational Biology*, 13(7):1276–1288, 2006.

[4] Lincong Wang and Bruce Randall Donald. Analysis of a Systematic Search-Based Algorithm for Determining Protein Backbone Structure from a Minimal Number of Residual Dipolar Couplings. In *Proceedings of the 2004 IEEE Computational Systems Bioinformatics Conference (CSB04), Stanford CA*, pages 319–330, 2004.

4 Installation

Since RDC-ANALYTIC is written in Java, it requires JDK 1.6. Henceforth, it is assumed that JDK 1.6 has already been installed. To install RDC-ANALYTIC

1. Unpack the tar file in a directory of your choice. Then go to the (sub)directory that contains the directory structure shown in Figure 1.

2. The Java files are in the directory `./src/analytic/`, and the class files (after compilation) will be in the following directory: `./analytic/`. To compile the Java files type the following command:

   ```
   javac -d .  -classpath ./javax/vecmath:/Jampack/Jampack:.  ./src/analytic/*.java
   ```

3. This completes installation, and RDC-ANALYTIC is ready for use.

4. To run the program type the following command:

   ```
   java analytic/RDCAnalytic
   ```

5 Configurations, Input and Output

The inputs to RDC-ANALYTIC are (1) N-H^N and C^α-H^α RDCs in one medium, and (2) secondary structure element specification (helix or sheet along with their boundaries), and (3) a sparse set of NOEs to pack the β-strands into β-sheets, and to pack the secondary structure elements (helix and/or sheet) to obtain the global fold. While packing β-strands into β-sheets is done by RDC-ANALYTIC, for packing β-sheets and α-helices after they are computed, we recommend using the PACKER module of RDC-PANDA[1]. RDC-ANALYTIC also comes with a packer which can be invoked by typing

```
java analytic/Packer <arguments>
```

Help on its usage can be found by typing

```
java analytic/Packer -help
```

5.1 File Organization of RDC-ANALYTIC

The directory structure of RDC-ANALYTIC is shown in Figure 1.

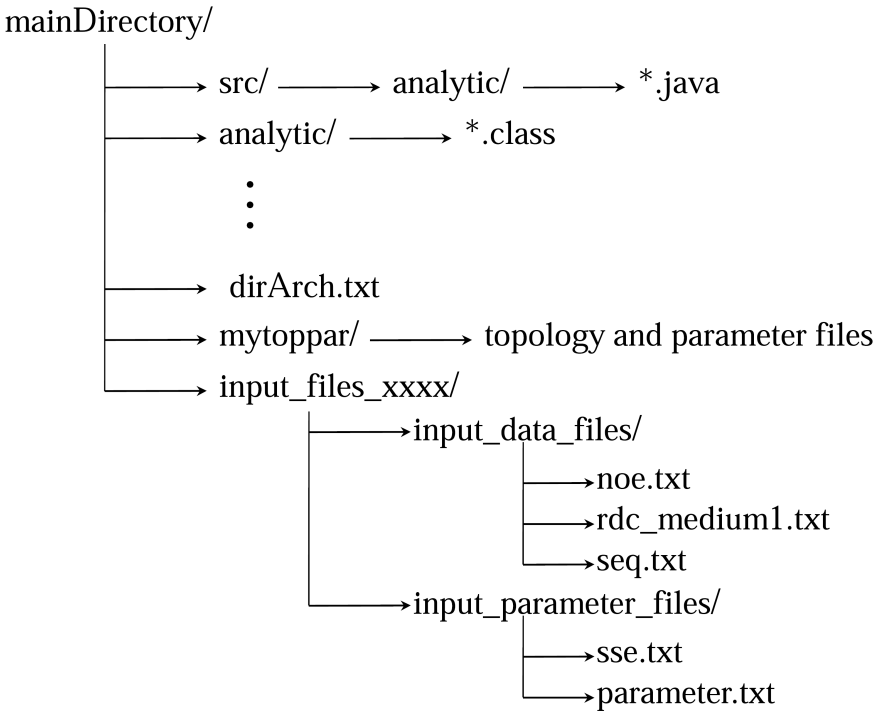

Figure 1: Directory structure of RDC-ANALYTIC.

Henceforth, we denote the main working directory (`mainDirectory` in Figure 1) for RDC-ANALYTIC by a period (`.`). The Java source files are located in the folder `./src/analytic/`. The Java binary class files are in the folder `./analytic/`. The file `dirArch.txt` specifies the above directory architecture, and the program uses this file to understand the input file organization. For a particular protein, when the input directory is to be specified, the user just needs to make modification (edit `xxxx` part) in one line in this file:

`inputDirectory: input_files_xxxx`

where `xxxx` is the user-specified part of the name of the input directory (see also Figure 1). This instructs RDC-ANALYTIC to read from the correct set of input files.

The directory `./mytoppar/` contains the information about the geometry of the protein, the RDC gyromagnetic ratios, and a set of constants for the program. The files in this directory need not be changed.

The directory `./input_files_xxxx/` contains two subdirectories (see Figure 1). The (sub)directory `input_data_files/` contains three files `noe.txt, rdc_medium1.txt,` and `seq.txt`, which specify the input data. The (sub)directory `input_parameter_files/`) contains the files `parameter.txt` and `sse.txt`, which specify the user-supplied parameters and the secondary structure specifications. The formats of the files are described below.

5.2 Input Format

noe.txt

This file contains the NOEs to be used to pack the β-strands into β-sheets. Only a sparse set of unambiguous NOEs (about 4 NOEs per pair or strands) are needed to pack the β-strands into β-sheets. These NOEs can be obtained from chemical shift analysis of small proteins, or Isoleucine-Leucine-Valine methyl labeling strategies used for larger proteins. Although backbone NOEs are preferred, RDC-ANALYTIC can use side-chain NOEs and can still pack the strands reliably. XPLOR format for NOEs is used, e.g.,

```
// example NOE
assign ((resid 5 and name HG2#)) ((resid 67 and name HN)) 3.555 3.555 0.876 !
```

The program requires that the interacting proton names conform with the latest PDB naming convention. A line comment in the file `noe.txt` starts with `//` as shown above.

rdc_medium1.txt

This file contains the RDCs in XPLOR format, e.g.,

```
assign ( resid 500 and name OO)
       ( resid 500 and name Z)
       ( resid 500 and name X)
       ( resid 500 and name Y)
       ( resid 15 and name N)
       ( resid 15 and name HN) -10.5000 0.0000 0.0000
```

The N-H^N and C^α-H^α RDCs are read from this file.

seq.txt

This file specifies the amino acid sequence of the protein in the format

`residueNumber  threeLetterIUPACAminoAcidName`.

sse.txt

This file specifies the secondary structure boundaries and types, and the topology of β-sheets. To specify a helix between the residue 23 and 33 one needs to type the following

```
@helix(23, 33, 4)
```

The tag `@helix` is used to specify a helix, and the boundaries are specified as shown above. Each secondary structure element (helix or strand) is given a user-defined unique serial number (for example, the number 4 above). Similarly, to specify a strand between residue 2 to 7 with user-defined unique serial number 3 one needs to type the following:

```
@strand(2, 7, 3)
```

To specify the helix boundaries that will be used to bootstrap the alignment tensor, one needs to type the following:

```
@computeAlignmentTensorUsing(@helix(23, 33, 4))
```

It is required that when specifying the helix and strand boundaries, the N-H^N RDC for the first residue of the helix/strand must be present.

A β-sheet between three strands with serial number 2, 3 and 5 with 2 and 3 being antiparalled, 2 and 5 being parallel with 2 being the middle strand (this information can be obtained by analyzing the unambiguous NOEs obtained from chemicah shifts or from ILV-Methyl selective labeling) is specified by the following line:

`@sheet( (3, 2, antiparallel) (2, 5, parallel))`

Since the strands are computed in the order specified in the β-sheet specification from left-to-right (e.g., in the example above the computation order of strands is 3, 2 and 5) care must be taken to specify the β-sheet so that the left-to-right evaluation order is meaningful. That is, in the example, for the strand 5 to be computed the pre-condition is that the strands with serial numbers 3 and then 2 have already been computed. For a β-sheet with 5 strands numbered from 11 to 15 (say), with strand $i+1$ being adjacent to strand i (for $11 \leq i \leq 14$), and strand 12 being the longest one, we recommend to specify the sheet in a way similar to shown below.

`@sheet((12, 13, parallel) (12, 11, antiparallel) (13, 14, antiparallel), (14, 15, antiparallel))`

Here `parallel` and `antiparallel` labels depend on the topology of the sheet. Also, we recommend to start from a strand somewhere in the middle of the sheet and grow the sheet by adding strands on either side. A reason for such a recommendation is that the set of NOEs used for packing at times don't provide enough constraints to yield a reasonable β-sheet, so starting from the middle may help in such situations. Finally, multiple sheets can be specified with `@sheet` tag multiple times (one sheet per line).

parameter.txt

The format of `parameter.txt` is given below.

```
// Notes:
// (1) numberOfSearchTrees can be set to 'auto' instead of a positive integer
// (2) RDC scaling factor is also specified in this file

@parametersForAlignmentTensorComputation {
    weightForRmsDihedralDeviationTerm = 8.0
    numberOfOptimizationsOfPP1        = 2 // an integer
    numberOfSearchTrees               = auto // an integer
}

@parametersForHelixComputation {
    weightForRmsDihedralDeviationTerm = 8.0
    numberOfOptimizationsOfPP1        = 2 // an integer
    numberOfSearchTrees               = auto // an integer
    gridResolutionInDegrees           = 1.0
}

@parametersForStrandComputation {
    weightForRmsDihedralDeviationTerm = 4.0
    numberOfOptimizationsOfPP1        = 2 // an integer
    numberOfSearchTrees               = auto // an integer
```

```
    gridResolutionInDegrees           = 2.0
}

// **RDC Scaling Factors**
// The flag scaleRdcTo can be set to one of the values from the following set:
// {scaled, CA_HA, N_HN}. It sets the values of the prefactors (Dmax) for
// the different types of RDCs measured. If the data has already been scaled,
// then use the flag 'scaled'. If the data are to be scaled wrt. CA_HA then set
// the flag to 'CA_HA', and if the data are to be scaled wrt. N_HN then set the
// flag to 'N_HN'. We recommend to use scaled RDCs for our program or to scale
// RDCs wrt. CA_HA.

scaleRdcTo CA_HA
```

The scaling method used for RDCs need to be specified as mentioned above. When the `numberOfSearchTrees` is set to `auto` the program picks a suitable integer on behalf of the user; otherwise, the user can supply a positive integer in the place of `auto`. The values of other parameters can be changed, but we suggest not to change them as they can have significant effect on the performance of the program and accuracy of the output.

In addition, the user can supply a few optional parameters. We recommend that these parameters be used with proper understanding of their effect on the computation, and when there is an absolute need to change their default values. It is not necessary to specify (or to provide a value for) these parameters, as the absence of specification of these parameters is handled automatically by the program. We list the parameters below.

- **testIfRdcsExistForBoundaryResiduesOfSse:** The usage of this boolean flag is:

  `testIfRdcsExistForBoundaryResiduesOfSse true/false`

  The default value for this flag is `true`. If set to `true` then it checks for if there exists RDCs for the boundary residues; otherwise, the checking is skipped allowing the user specify a SSE boundary for which there need not be any data present. The user must be careful to make such a decision.

- **enforceRdcErrorIntervalEarly:** The usage of this boolean flag is:

  `enforceRdcErrorIntervalEarly true/false`

  The default value for this flag is `false`. If set to `true` then it enforces a narrow RDC error interval; otherwise, it allows the program to choose the value appropriately. The user must be careful to make such a decision. If there are long helices with lots of missing RDCs, then the user may try setting this flag to `true`.

- **computeAlignmentTensor:** The usage of this boolean flag is:

  `computeAlignmentTensor true/false`

  The default value of this flag is `true`. If set to `true` and if the alignment tensor specification is provided in the file `sse.txt` then the program estimates the alignment tensor. If this flag is set to `false` then the user must provide the values for the alignment tensors in the file

parameter.txt. For example, the alignment tensor specification in the file parameter.txt will look like

```
...
...
...
computeAlignmentTensor false
Syy 14.3
Szz 24.9
...
...
```

where the alignment tensor components are Syy and Szz, respectively. Note that if the flag computeAlignmentTensor is set to true then even if Syy and Szz are specified, the program discards these values and computes the alignment tensor.

5.3 Output Format

All computed secondary structure elements are stored in the file bestSetOfFragments.txt. The file bestFragment.txt stores the best conformation of the last secondary structure element (SSE) computed. Also, for each helix or β-sheet for convenience separate files are written, e.g., for ubiquitin helix1.pdb and sheet1.pdb are generated. The file at.pdb holds the coordinates of the helix computed during the alignment tensor estimation. The file saupeList.txt contains the alignment tensor components. In addition, two more files that are generated in the working directory, viz., the file mostRecentFragment.pdb that stores the most recent solution fragment and the file phiPsiSolutionSets.txt that stores the phi-psi values and a solution counter for the last SSE computed. These two files are simple log files generated as side-effect, therefore, can be discarded.

The output files of interest are helix*.pdb and sheet*.pdb which are then supplied to the PACKER [1] module along with a sparse set of NOEs for packing to obtain the global fold.

6 Examples

This distribution comes with examples of how to prepare the input files and run RDC-ANALYTIC on two proteins, namely, ubiquitin and FF2. The input files are prepared as explained in Section 5.2. ./input_files_1d3z/ contains the input files for ubiquitin and ./input_files_2kiq/ contains the input files for FF2. Before running the test for ubiquitin, the input directory input_files_1d3z must be specified in dirArch.txt as explained in Section 5.1 so that correct set of input files are read by RDC-ANALYTIC. Similarly, before running the test for FF2, the input directory input_files_2kiq must be specified in dirArch.txt. Since RDC-ANALYTIC is fully automated, all you need is to type the following command:

```
java analytic/RDCAnalytic
```

to run RDC-ANALYTIC.