The manual to Ducque

Jérôme Rihon[1,*]

[1]KU Leuven, Rega Institute for Medical Research, Medicinal Chemistry, Herestraat 49 - Box 1041, 3000 Leuven, Belgium

[*]*Corresponding author, maintainer*

69

Contents

Supervisor : prof. dr. Eveline Lescrinier
Co-promotor : prof. dr. Vitor Bernardes Pinheiro
Co-promotor : prof. dr. Matheus Froeyen

dr. Rinaldo Wander Montalvao for his guidance on the fundamentals of linear algebra.
dr. Charles-Alexandre Mattelaer for his guidance on Quantum Mechanics and without his experimental work, Ducque could have never been conceived.

1 Introduction

> There are no mistakes, only happy little accidents.
>
> *Bob Ross*

A software for the purpose of building native and synthetic nucleic acid duplexes.

Ducque (IPA : /dʌk/) stands for :

D Acronyms
U Are
C Rather
Q
U
E Tedious

The name was inspired much in the same way that the ORCA Quantum Mechanics package[1,2] was named. At the time of development, I had different name for Ducque. During one of my evaluation moments, one of the members of my jury pointed out that with that name, I would probably not get enough traction, since Google searches gaves millions of results. Given that fact, I tried coming up with a new name for the model builder.
Weeks later, as I was working, my hat that was resting on my desk in front of me caught my eye. You see, my hat has a tiny duck embroided on the front. With this idea, I *"researched"* ways to rewrite the word and get as little search engine results as possible, without degenerating the word itself and keeping it phonetically somewhat correct. Thusly, Ducque was born (a second time).

1.1 Modules

Ducque has five modules the user can access.

1. The **build** module, which is the primary module for to use. This will build us the duplex structures we want to use for further
2. The **transmute** will be used, together with an elaborate input, to convert the ‘pdb‘ file to a suitable input for Ducque to use as a building block. The appropriate format has been settled to be the very simple ‘json‘ format.
3. For testing purposes or just to be able to generate randomised sequences, there is a **randomise** module, which generates a randomised duplex structure based on a set of given inputs.
4. To be able to convert ‘xyz‘ formatted molecule structures to a ‘pdb‘ format, I have included a **xyz_pdb** module. Since I am an avid ORCA user and the ‘xyz‘ format is often outputted, I wanted to make it more managable to format a ‘pdb‘ file for myself during development.
5. To have clickable objects for the standard non-terminal user, there is a simple **gui** that can be used to employ the following modules : build, transmute and randomise. The reason that the xyz_pdb is not included is because I did not find a nice way to design a simple gui. Secondly, other QM programs might have different outputs of molecule structures, the ‘xyz‘ format can be rather niche and therefor not worth to effort to design a good gui for. Chimera and PyMol can definitely convert from one format to the other if one cannot work through the CLI.

1.2 Package structure

path/to/Ducque/

→ **bin/** : contains executable to run Ducque
→ **docs/**: contains this pdf and its LaTeXsrc
→ **json/**: contains building blocks requires by Ducque to build duplex sequences. The name for the json file is derived from the inputs of the --transmute *INPUTFILE*. The filename is there for classification and is thereby non-trivial.
→ **pdb/** : *delete this dir*
→ **puckerdata/** : contains all optimised xyz and pdb files, needed by --transmute to be converted to json
→ **src/** : contains all the Ducque source code
→ **tests/**: testing module
→ **transmute/**: contains the --transmute input files, just as an example. The name of the file itself is trivial.
→ **xyz/** : contains the --xyz_pdb input files, just as an example. The name of the file itself is trivial.
→ LICENSE : GPL2 License to ensure FOSS!
→ README.md : github readme
→ setup.sh : required to make Ducque executable by default

2 Usage

2.1 Build

```
$ Ducque --build INPUTFILE

The inputfile is read in and the sequence is built accordingly.
At any given time, there are two (2) flags in total that should be involved in the Ducque.

--sequence SEQUENCE
    Only valid input in the file just a string of the required nucleotides (comma-delimited).
        Example: --sequence dT, dC, dA, dA, dC, dG, dG, dT, dA

--complement COMPLEMENT
    The complement flag denotes the structure of the complementary strand
    A list of nucleotides is also a valid input, if one wants a specific complementary strand.
        Example: --complement homo
        Example: --complement dT, dA, dC, dC, dG, dT, dT, dG, dA

--pdbname PDBNAME
    To prompt the name of the pdbfile for the outputted structure.
    If none is given, this defaults to the name of the queried inputfile.
```

2.2 Transmute

```
$ Ducque --transmute INPUTFILE

The inputfile is read in and the json file is formatted accordingly.
There are five (5) flags needed to convert from `pdb` to a `json` file format.

--pdb PDB
    The name of the file of the structure you want to convert to json
        Example: --pdb dna_A.pdb

--chemistry ID
    The chemistry that defines the given nucleo(s)(t)ide
        Example: --chemistry DNA

--comformation CONFORMATION
    The conformation that denotes the nucleic acid.
    Used to name the output .json file.
    Allows multiple conformers for a given chemistry.
    Used when building the complementary strand.
    The complementary strand is then fitted onto the leading strand.
        Example: --conformation 2endo
        Example: --conformation 1-4boat

--moiety MOIETY
    The moiety that the structure defines. Should either be "nucleoside" or "linker"
        Example: --moiety nucleoside

--bondangles ALPHA, BETA, GAMMA, DELTA, EPSILON, ZETA, CHI
    The bond angles involved in the backbone and the anomeric carbon, DEGREES
    Comma-delimited string.
        Example: --bondangles 101.407, 118.980, 110.017, 115.788, 111.943, 119.045, 126.013

--dihedrals ALPHA, BETA, GAMMA, DELTA, EPSILON, ZETA, CHI
    The dihedrals involved in the backbone and the anomeric carbon, DEGREES
    Comma-delimited string.
        Example: --dihedrals -39.246, -151.431, 30.929, 156.517, 159.171, -98.922, -99.315
```

Standard JSON structure of the Ducque input

```
    {
pdb_properties : { Coordinates : [X, Y, Z] ,
                       Shape : (rows, columns) ,
                       Atoms : [O5', C5', H5'1, H5'2, C4' ··· , O3'] ,
                       Symbol : [O, C, H, H, C  ··· , O ]
                      }

identity: [ Chemistry, Abbreviation, Residue name, Base ]

angles : { dihedrals : { Alpha : X,
                             Beta : X,
                             Gamma : X,
                             Delta : X,
                             Epsilon : X,
                             Zeta : X,
                             Chi : X
                             }
```

```
        }

        { bond_angles : { Alpha : X,
                          Beta : X,
                          Gamma : X,
                          Delta : X,
                          Epsilon : X,
                          Chi : X
                          }
        }
}
```

2.3 Randomise

```
$ Ducque --randomise INPUTFILE

The inputfile is read in and --Ducque inputfile is generated.
Only two (2) flags should be queried for the randomisation.
    Using --length and --sequence are mutually exclusive.

--chemistry CHEMISTRY
    The chemistry that defines the prompted nucleotide's
        Example: --chemistry DNA
        Example: --chemistry DNA, RNA, TNA

--length LENGTH
    The length, in amount of nucleotides, of the sequence the user wants to generate.
        Example: --length 30

--sequence SEQUENCE
    Provide the file with a sequence. Only the bases are required.
    The values are comma-delimited :
        Example: --sequence A, C, T, G, G, A, A, T, C, A

--complement COMPLEMENT (Optional)
    Fills in the `--complement` flag in the input file to build.
    See the '--complement' flag of the '--Ducque' function.
```

2.4 XYZ to PDB

```
$ Ducque --xyz_pdb INPUTFILE
The inputfile is read, the xyz file used as an input to output a well formatted pdb.

There are three (3) flags involved in converting a `xyz` coordinate file to a `pdb` file.
--xyz XYZ
    The name of the file of the molecule you want to convert to pdb
       Example --xyz dna_2endo.xyz

--residue RESIDUE
    The identifier of the nucleo(s)(t)ide.
    See the PDB format %link-to-pdb-format.
    NB : Ducque does not allow custom nucleic acid chemistries with a RESIDUE unequal to three!
        (in this example, dXA is equal to deoxy Xylose nucleic acid with an adenine base)
        Example: --atomID dXY

--atomname_list ATOMNAME_LIST
    The ordered list of atoms that belong in the 'Atom name' column in a pdb file.
    See the PDB format %link-to-pdb-format.
    The order needs to so that it follows the order of the atoms from the xyz file.
    DISCLAIMER : the responsability is with the end-user to see everything is correct.
        Example: --atomname_list O5', C5', H5'1, H5'2, C4' ··· , O3'
```

2.5 Graphic User Interface (GUI)

2.6 User implementation of custom nucleic acids

Before adding a new chemistry
In $TRANSMUTETOOLS :

1. Add to the $NUCLEOSIDEDICT the type of chemistry it is. The key should be in all caps.
2. Add to the $LINKERDICT the type of linker it corresponds with. The key should be in all caps.
3. This only serves the purpose of identifying the json file when opening it as the user. This information is not used in the generation of nucleic acid duplexes.

Before generating a duplex with the new chemistry
In $REPOSITORY :

1. Add to the $CHEMCODEX the most stable conformation of the chemistry you're adding to the library.
2. Add to the $CONFCODEX all the conformations that you have at your disposal of the chemistry you're adding to the library. The key in both these abbreviated name of the nucleic acid chemistry.
3. Add to the $BACKBONECODEX the sugar linker backbone of the chemistry you're adding to the library.

3 Installation

3.1 Set-up

To access the Ducque software from anywhere on your machine, add the following line to your `$HOME/.bashrc` . Where path/to/program is the path to where you've installed Ducque.
Use the `$pwd` command inside the Ducque directory if you're unsure. Depending on your `python3` version, you might have to install tkinter separately.

```
export PATH=$PATH:path/to/program/Ducque/bin # in the $HOME/.bashrc
```

```
$ cd path/to/Ducque    # Go to the Ducque directory, where you installed it
$ chmod +x setup.sh    # Give permission to run as an executable script
$ ./setup.sh           # Run the setup.sh script
```

3.2 Environment

```
# Using the pip package manager
$ pip install numpy
$ pip install scipy

# Using the conda package manager
$ conda install -c numpy
$ conda install -c scipy

# Check if you have tkinter installed :
 $ python3 -m tkinter # if not succesful ...
 $ sudo apt-get install python3-tk # install this
```

References

[1] Frank Neese, Frank Wennmohs, Ute Becker, and Christoph Riplinger. The ORCA quantum chemistry program package. *J. Chem. Phys.*, 152(22):224108, jun 2020.

[2] Frank Neese. Software update: The ORCA program system—version 5.0. *WIREs Computational Molecular Science*, 12(5), mar 2022.