# User's Manual for *fg_simulations v0.99*

Simulating Nucleocytoplasmic Transport
using the Integrative Modeling Platform (IMP)

Shaked Rashkovits shaked.rashkovits@mail.huji.ac.il
Roi Eliasian roi.eliasian@mail.huji.ac.il
Barak Raveh barak.raveh@mail.huji.ac.il
***Last updated:*** *Feb 14, 2025*

## Table of Contents

# Key References

**Nucleocytoplasmic transport through the full NPC:**
Raveh B*, Eliasian R, Rashkovits S, Russel D, Hayama R, Sparks SE, Singh D, Lim R, Villa E, Rout MP*, Cowburn D*, Sali A* (2025) Integrative mapping reveals molecular features underlying the mechanism of nucleocytoplasmic transport. *In review*.
**Simplified cylinder model of passive diffusion through the NPC:**
Timney B*, Raveh B*, Mironska R, Trivedi JM, Kim SJ, Russel D, Wente SR, Sali A*, Rout MP* (2016) Simple rules for passive diffusion through the nuclear pore complex. *Journal of Cell Biology* 215 (1): 57−76.

# Preliminary requirements

- IMP (Integrative modeling platform)
    - General installation instructions (including using `Conda` or `Google Colab`):
      https://integrativemodeling.org/nightly/doc/manual/installation.html
      https://integrativemodeling.org/download.html
    - Source files for IMP: https://github.com/salilab/imp/
      (we refer to this folder as **`$IMP_GITHUB`**)
    - Source files of the MPC transport module of IMP, including the source code for fg_simulations:
      https://github.com/salilab/npctransport/tree/develop/
      (we refer to this folder as **`$NPCTRANSPORT_GITHUB`**)
- **For source installations**, the manual assumes the following constants
    - **`$IMP_FOLDER`** - The IMP installation root folder. For example[1]:
      `IMP_FOLDER=/cs/labs/ravehb/ravehb/imp/fast_conda/`
    - **`$IMP`** - The IMP setup script. For example:
      `IMP=$IMP_FOLDER/setup_environment.sh`
    - **`$IMP_SRC`** - The IMP source repository. For example:
      `IMP_SRC=/cs/labs/ravehb/ravehb/imp/repository_editing/`
    - **`$FG_SIMULATION`** - running the FG simulation binary
      `FG_SIMULATION="$IMP $IMP_FOLDER/bin/fg_simulation"`
    - **`$IMP_PYTHON`**
      A Python installation from which IMP was built[2] , initialized using the IMP setup

[1] This is the actuall installation folder for users on Raveh lab HUJI servers
[2] For users on Raveh lab HUJI computers using the default conda installation and zsh, you can run:

```
. "/cs/labs/ravehb/ravehb/miniconda3_v2/etc/profile.d/conda.sh"
conda activate py312_imp_build
```

script. For example:
`IMP_PYTHON=$IMP python`

- For **non-source installations**:
  - Ignore `$IMP_FOLDER` and `$IMP` or set them to `""`, e.g.:
    `IMP_FOLDER=""`
    `IMP=""`
  - `FG_SIMULATION=fg_simulation`
  - Anything in `$IMP_SRC` can be found in the corresponding folder in **`$IMP_GITHUB`**
  - Set **`$IMP_PYTHON`** to the Python installation you used for installing IMP on conda/colab/etc. (possibly, simply `IMP_PYTHON=python` if you are running from the correct environment). For example:
    `IMP_PYTHON=$IMP python`

# Basic Usage

## Input files

- **Option 1:** A new configuration file
  A new simulation requires a Google protobuf configuration file `<cfg_name>.pb`, as specified in the npctransport.proto file:
  `$NPCTRANSPORT_GITHUB/data/npctransport.proto`
  For more information, see Creating a configuration file below.
- **Option 2:** restarting from a previous run
  The simulations can be restarted from the output protobuf file of a previous run using the `-restart` flag, e.g. output.pb (see *Output files* below)

## Output files

1. Output file in Google Protobuf format (default name - `output.pb`)
   Fields are as specified in the `output` protobuf "message" in `npctransport.proto` file. It also can be viewed in text format using a convenience python script:
   `$IMP_PYTHON $IMP_SRC/npctransport/utility/npc_show_output.py`
2. An HDF5 file, e.g. `output.hdf5`
   This file contains information about the distribution in space of different particles in the simulation. (a three-dimensional matrix that describes how many times each molecule visited each voxel in 3D space).
3. Trajectory file (default name - `movie.rmf`)
   The snapshots from the simulation trajectories are saved in movie.rmf in the Rich Molecular Format. Files in this format can be viewed in molecular viewers such as `Chimera` and `ChimeraX`.

4. Final frame file (default `final_conformations.rmf`)
   This file includes the last frame in the simulation trajectory in the same format as the trajectory file.

## Running the simulation

After following all the above prerequisites and preparing an input configuration file (e.g. *`config/config.pb`* in this example), run the simulation as follows:

### A typical full NPC simulation

```
$FG_SIMULATION --configuration config/config.pb -output
Output/output.pb --short_init_factor 0.5 --conformations
Output/movie.rmf --final_conformations Output/final_conformations.rmf
--random_seed $seed
```

### A typical simplified NPC simulation

```
$FG_SIMULATION --configuration config/config.pb --cylinder_nlayers 4
--output Output/output.pb --short_init_factor 0.5 --short_sim_factor
1.00 --conformations Output/movie.rmf --final_conformations
Output/final_conformations.rmf --random_seed $random_seed
```

### Restarting a simulation from a previous run

```
$FG_SIMULATION --restart Output/prev_output.pb --cylinder_nlayers 4
--output Output/new_output.pb --conformations Output/new_movie.rmf
--final_conformations Output/new_final_conformations.rmf
```

***Note:*** *do not use the* `-configuration` *flag when restarting a simulation.*

# Command-line arguments

Run `fg_simulation --help` to retrieve information on command-line arguments.

### Common flags

| Argument | Description |
|---|---|
| `--configuration arg (= configuration.pb)` | input configuration file in protobuf format; do not use this flag along with the `-restart` flag [default: %default] |
| `--output arg (= output.pb)` | output assignments and statistics file in Google's protobuf format, recording the assignments being executed |
| `--conformations arg` | RMF file for recording the conformations along the simulation, e.g. `movie.rmf`. |
| `--final_conformations arg (=final_conformations.rmf)` | RMF file for recording the initial and final conformations [default: %default] |
| `--restart arg` | output file from a previous run, from which to restart this run instead of using the configuration file (also initializing the final coordinates from this previous run, if they exist in the output file) **Note:** Do not use this flag along with the `.-configuration` flag |
| `--cylinder_nlayers arg (=0)` | anchor FG nups to pore walls automatically, with specified number of FG layers (no cylinder anchoring if equals to 0). |
| `—random_seed` | randomize from a certain seed; default is an IMP-generated stochastic seed e.g. using boost::random_device() |
| `--short_init_factor arg (=1)` | Run an abbreviated version of system initialization, which takes the specified fraction of a full initialization (or more if >1.0) [default=1.0] Note: initialization does not run in restart mode unless --force_initialization_on_restart flag is specified explicitly |

| Argument | Description |
|---|---|
| `--short_sim_factor arg (=1)` | Run an abbreviated version of the simulation, which takes the specified fraction of a full simulation (or more if >1.0) [default=1.0] |
| `--initialize_only` | Run the initialization phase and then stop |
| `-h [ --help ]` | Show command line arguments and exit |
| `--version` | Show version info and exit. |
| `--help_advanced` | Show all command line arguments including advanced ones and exit. |
| `--verbose` | Print verbose information during run |
| `--log_level arg (=WARNING)` | The log level: "SILENT", "WARNING", "PROGRESS", "TERSE", "VERBOSE" |

## Advanced flags

| Argument | Description |
|---|---|
| `--no_save_rmf_to_output` | If true, save final rmf buffer to output file [default=false] |
| `--work_unit arg (=-1)` | The work unit. If parameter ranges are specified in the protobuf configuration file, the work unit specifies the parameter combination that will be used, modulo the total number of combination (e.g. if there are 3x10x5 parameter combination, work ids 0..149 will cover all of them, and igher work ids will be computed modulo 150). |
| `--show_number_of_work_units` | Show the total number of work units |
| `--restart_fgs_only arg` | output file of a previous run, from which to restart only the FGs (but not diffusers). The system then goes through equilibration with the new diffusers, but not the FGs. The new diffusers are still randomized by default. [default: %default] |

| | |
|---|---|
| `--remove_fgs_with_prefix arg` | Removes all fgs whose type begins with specified prefix from the simulation and from the output file. Note this forces statistics to be reset |
| `--init_rmffile arg` | [OBSOLETE - do not use this flag] |
| `--first_only` | Only do the first simulation iteration |
| `--force_initialization_on_restart` | Force re-initalization on restart, but without randomizing positions (essentially, relaxation from specified starting position) |
| `--no_save_restraints_to_rmf` | whether not to save restraints to rmf conformations and final rmf, if either is applicable |
| `--inflate_kap28` | whether to inflate kap28 to radius of 150 nm + double the box size + switch it to having 16 interaction sites |
| `--kap_interaction_k_factor arg (=1)` | increase kap interaction factor by said ratio relative to input configuration/restart file, and update to output file |
| `--surface_anchoring` | anchor FG nups to bottom of cube |

### Convenience script (HUJI)

For a quick start on HUJI servers, edit `sbatch_script.sh` file (`/cs/labs/ravehb/archive/littlem/npc_example/Figure5B_simulations/sbatch_script.sh` file) by only editing the configuration file name, log file name and the simulation name (all other arguments are initiallized to a default value).

# Creating a configuration file

To create a configuration file, we can use a python script, e.g. `make_cfg.py` , which could be modified to change the running parameters, e.g.:

```
$IMP_PYTHON ./make_cfg.py cfg_name.pb > cfg_name.txt
```

## Full yeast NPC (as in Raveh *et al.* 2025)

1. Download the script for creating the configuration file: https://github.cs.huji.ac.il/ravehb-lab/npctransport_util/blob/master/Scripts/load_whole_new_coarse_grained_v15.py[3]
2. Download a model of the NPC scaffold in RMF format: https://github.cs.huji.ac.il/ravehb-lab/npctransport_util/blob/master/Models/NPC2018/47-35_1spoke.rmf3[4]
    a. ()
3. Create a load.params file:

```
{
 "FGParams": {
   "default_FSFG":{
     "kap_k": 2.64
   },
   "default_GLFG":{
     "kap_k": 2.64
  }
 }
}
```

4. Create a configuration file using:

```
bash ${IMPPY} Scripts/load_whole_new_coarse_grained_v15.py
\
       --params_file load.params --advanced_include_Nup2 \
       Config/config.pb 47-35_1spoke.rmf3                    \
       1> Config/config.txt 2> Config/confi
```

[3] on Raveh lab computers:
`/cs/labs/ravehb/ravehb/Projects/Npctransport/npctransport_util/Scripts/load_whole_new_coarse_grained_v15.py`

[4] on Raveh lab computers - copy or soft link to:
`/cs/labs/ravehb/ravehb/Projects/Npctransport/npctransport_util/Models/NPC2018/47-35_1spoke.rmf3`

## Simplified NPC (e.g. cylinder)

### Edit configuration parameters

Start from an existing make_cfg.py file (e.g. `/cs/labs/ravehb/archive/littlem/npc_example/Figure5B_simulations/make_cfg.py` in HUJI) where all the parameters of the configuration are initialized to a default value. To edit the configuration parameters, we create a copy of the make_cfg file by the command: `cp make_cfg.py <new_file_name.py>`. Then we can edit the new file with any text editor we choose.

## Common configuration file parameters

More details can be found in the protobuf file scheme in
https://github.com/salilab/npctransport/blob/develop/data/npctransport.proto

| **Parameter name** | **Description** |
|---|---|
| *Kap_interaction_sites* | The amount of kaps link sites to fg repeats |
| *Tunnel radius* | radius of the hole in the nuclear membrane where the NPC is. In units of measure of Angstrom (10 ^ -10 meters). |
| *Interaction_range* | The minimum distance required to create an interaction between kap and nup. |
| *Simulation_time_ns* | time of the simulation in naneseconds |
| *Box_is_on* | a box area which no particle can come out of, due to a physical force exerted on it.<br>Options are: 0 - no box (not in use), 1 - box , 2- a sphere instead of a box |
| *Box_side* | the side of the box described above (The diameter if it is a sphere). |
| dump_interval_ns | The interval in nanoseconds at which a new frame is written to the trajectory .rmf file, if applicable.<br>*Notes:* |

| Parameter name | Description |
|---|---|
| | 1) *if this interval is low (high dump rate), it may affect the running time*<br>2) *If output_statistics_interval_ns is lower than dump_interval_ns, it will dominate the RMF output rate instead* |
| output_statistics_interval_ns | The interval in nanoseconds at which the output file (e.g. output.pb) is being rewritten, and at which order parameters for various statistics are being updated. Note if this happens too often, it may affect the running time |
| statistics_interval_ns | The interval at nanoseconds at which statistics are being monitored during the run (note this does not result in a file output operation) |
| *is_xyz_hist_stats:* | whether to add XYZ statistics to the simulation in an .hdf5 file (default used to be false until August 2023, this may change to true; also by default half of each box's dimension is cropped) |

## Common configuration file utility functions (e.g. in this script)

`add_fg_type()`

Adds the Nup we use in the simulation:

- config- the configuration object we are updating
- Type name - a string indicating the name of this species of FG repeats (e.g. "Nup49" or "fg0"). If none is specified, fg_simulation will call it "fg0", "fg1", etc. by default.
- Number_of_beads- each fg repeat is represented by a string of beads. This parameter determines the number of beads in each fg repeat (and by that it's length). Typically, each bead represents 20 amino acids (though this is a user's choice)
- Number - number of distinct FG repeat chains (= copies)
- Radius - radius of a single bead in the chain of FG repeats in angstroms. A typical value is 8 angstroms (but may vary considerably).
- Interactions - the number of interaction sites placed on the surface of each FG bead, representing a single FG binding motif; if i=-1 then FG bead is itself a site. In Chimera, these are typically visualized as red spheres on the surface of FG beads (see Visualization below).

- Rest_length_factor - a multiplier for the default resting length for the spring connecting consecutive FG bead centers. By default, the resting length is equal to the sum of the beads' radii. A typical value for the multiplier is 1.7 (but may vary considerably).

`add_float_type()`
Adds spherical particles representing diffusing ("floating") molecules:

- config- the configuration object we are updating
- Number - the amount of this particle to add
- Radius - radius of the particle
- Type name- a string indicating the name of molecules of this type (e.g. "inert20" or "kap30").
- Interactions - the number of interaction sites placed on the surface of each spherical particle, representing e.g. interaction sites on the surface of molecules.

`add_interaction()`
Adds an interaction (attraction) between particles of two different types in the simulation:

- Name0 - the first type of interacting molecules (see type_name in add_fg_type and add_float_type). Note this could also indicate a part of a molecular chain in advanced use.
- Name1- the second type of interaction molecules
- Interaction_k - the force of attraction between the two molecules. Measured in units of Energy/Angstrom$^2$
- Interaction_range -the minimum distance required to create an interaction between the particles.

**All length parameters are specified in units of measure of Ångstroms ($10^{-10}$ meters), and all the time parameters are specified in nanoseconds ($10^{-9}$ seconds).

For more optional parameters, check out the `npctransport.proto` file.

**Create a configuration file:**
Finally, to create the configuration file, run:

```
$IMP_PYTHON <new_file_name.py> <config file name.pb>
```

# Output analysis

## Computing spatial densities of various molecules

See `get_xyz_stats_v5.py` in npctransport_util/Scripts/ (in the lab GitHub https://github.cs.huji.ac.il/ravehb-lab/npctransport_util)

Setting "`config.is_xyz_hist_stats=1`" in the configuration file allows the output of voxel based spatial data. The space is divided into voxels, each of size "`config.xyz_stats_voxel_size_a`" (default=10). A statistical frame is specified by the "`statistics_interval_ns`" configuration parameter. So if we ran for 10,000 ns and each statistical frame takes 0.1 ns, we would have 100,000 statistical frames. During each of those statistical frames, we check, for each molecular type (i.e. FG repeats, passive molecules or NTR's) whether the center of a molecule was contained in any voxel. In that case we increase the count by 1. In the end, we are left with a 3d array for each molecular type, in which the entry (XYZ) is the amount of statistical frames during the whole simulation during which the molecule was in that voxel.
These arrays are then outputted in an HDF5 file that contains a table ("group") with the 3D voxel map for each type.

## Computing transport rates of various molecules

see `transport_stats_mpi.py` in npctransport_util/Scripts/ (in the lab GitHub https://github.cs.huji.ac.il/ravehb-lab/npctransport_util)
The following command should hopefully work:

```
$IMP python3 <script-folder>/transport_stats_mpi.py
$OUTPUT_FILES_PREFIX $WARMUP_TIME_IN_SECONDS
```

- `$OUTPUT_FILES_PREFIX` is a prefix for all the output files from independent runs that will be analyzed simultaneously, e.g. if it is equal to "`Output/output`", than "`Output/output*.pb`" will be analized
- `$WARMUP_TIME_IN_SECONDS` is the simulation time on which we skip, because we consider it to be non-representative statistically. For example, if it is equal to "`5e-6`", then we skip the first 5 microseconds of the simulation. Why is it needed? In the first few microseconds of the simulation, the system is still stabilizing and its statistics are not sufficiently reliable (this could be tested directly by comparing whether the statistics are sensitive to this parameter). Empirically, in the past we found out it's a good idea to skip over the first few microseconds, even 10 or 20 microeconds. But for experimentation, you can choose a lower number.
- *MPI (multiprocess interface):* Optionally, we can use this script with the MPI protocol for parallel computing, in order to analyze many files in parallel on multiple processors. Assuming one learned how to use MPI and installed the schwimmbad package (https://github.com/adrn/schwimmbad/), it used to work using something like this:

mpiexec -np $NSLOTS --mca btl ^openib $IMP python3 Scripts/transport_stats_mpi.py $OUTPUT_FILES_PREFIX $WARMUP_TIME_IN_SECONDS

## Visualizing trajectory files in Chimera

- There is a script in `<npctransport-source-code>/utility/add_cylinder` (--help, only from input_rmf is relevant)
  This script is adding a cylinder (a line) between the beads for a clearer vision of the file on Chimera.
  The 3 first arguments are most important:
  1. `--input_rmf arg`: The input RMF file (for example movie.rmf)
  2. `--output_rmf arg`: The output RMF file in which to add the cylinders.
  3. `--ref_output arg`: reference output file from which info e.g. FG nup types can be extracted (for example the output.pb file)
     To run this script, provide these 3 arguments. For information about more arguments option, run `add_cylinders.py --help` (notice only arguments from input_rmf and on are relevant)

- for visualization of the file, you can open it with Chimera (downland link: https://www.cgl.ucsf.edu/chimera/download.html)

Some of the objects that can be viewed on Chimera animations:

- FG Nups: you can see the different FG nups (FG nucleoporins proteins). Their names and parameters are defined in the configuration file.
  In our simulation, we only use one type of nup, with 4 layers (that's what the --cylinder_nlayers argument represents).
- "Obstacles": the scaffold of the NPC. A collection of hundreds of proteins creating the pore scaffold within the nuclear membrane. Their names and parameters are also defined in the configuration file
- Inert molecules: represent proteins, that are inside a box area whose dimensions are determined when running the simulation on the configuration file (box_side, box_is_on). These are the particles for which we want to measure their rate of passage through the NPC, and actually the distribution in their space
- Kaps (karyopherins): represent NTRs (nuclear transport receptors), particles that mediate the interaction between cargo molecules and FG repeats. NTRs include karyopherins (importins and exportins), NTF2 and various NTF2-like molecules.
- Bounding box geometry- No particle can come out of the box due to a physical force exerted on it.

On the chimera animation, choose which objects to show or hide, and by that to explore and analyze the distribution of the locations of the particles in space, interactions between them, the distance between them etc.

## High-Performance Computing (HPC) using Slurm

On HUJI computers, see https://wiki.cs.huji.ac.il/wiki/Slurm for more details.
One of the parameters of using slurm is for how long we need to use the servers. The servers will disconnect automatically at the end of the time.

# Specific setups

## Scanning a range of pore radii

To run the simulations multiple times and to investigate the relationship between NPC radius size and simulation results, we have created some convenience scripts.

### Preliminary Preparations

a) **Directory organization (recommendation):**
   - Logs- for log output files
   - Config- for configuration files
   - Production- for outputs files.
     Inside production, a separate folder should be created for each radius checked.
   - Test- if you want to try some test simulations.

b) **Think about the goals and choose which radii range to scan**

`In /cs/labs/ravehb/archive/shakedrash/passive_simulations` you will find:

- sbatch _script.sh - this is a source file of running a simulation with all the parameters needed. It gets radius size as user argument (sbatch the file by the command: sbatch _script.sh chosen_radius).
  The only parameter that needs to be changed here is --array which controls the number of times the simulation will run for a single radius. (array of jobs, for more information, see https://wiki.cs.huji.ac.il/wiki/Slurm).
- Create_script.sh - this is a script that prepares the files and folders required to run the simulation within a range of radii.
  It creates a configuration file suitable for each radius in the Config folder, and in addition creates a folder inside the Production folder for each radius that will be checked, with a copy of the executable file sbatch _script.sh with the name sbatch _script_radius.sh. After creating those, it's connected to Slurm.
  The radii tested can be easily modified by changing constants at the beginning of the file.
- Run_script.sh - this script is intended to run after running create_script.sh. this script submits the resulting jobs to a slurm server.
  Change the radius constants here to match the radius constants in the file create_script.sh.
- Analyse_script.sh - this script uses another script and creates .csv file with statistics of all simulations' output file in the Production folder, a separate one for each radius.

  **Similar files for active simulations exist in the active_simulations folder

After editing all the relevant files, it will automatically connect to slurm and then run run_script.sh. Then, for analysis, run Analyse_script.sh.

```
g.err
```