crease_ga

Zijie Wu, Arthi Jayaraman

Apr 22, 2024

GETTING STARTED

Last Updated: March 28, 2024

CHAPTER

ONE

1. INTRODUCTION TO CREASE

Members of Prof. Arthi Jayaraman's research lab have developed the **'Computational Reverse-Engineering Analysis for Scattering Experiments' (CREASE)** method to address these needs for alternate scattering analysis methods that are applicable to both conventional soft materials structures with existing analytical models and unconventional structures/chemistries that may not have good analytical models.

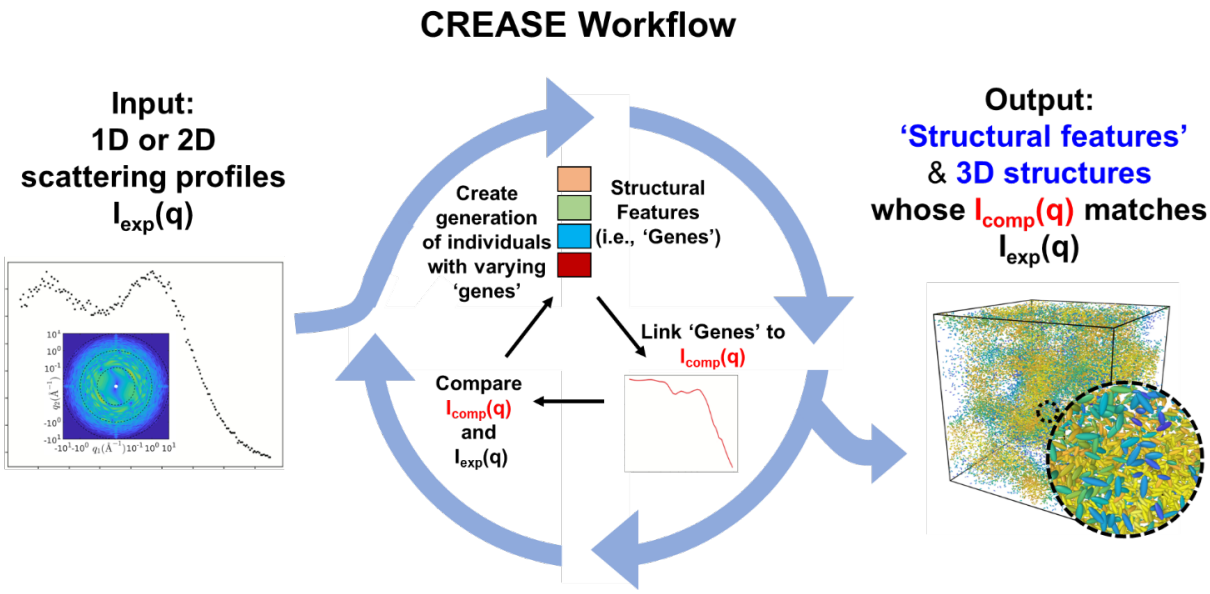

Fig. 1: Figure 1.: CREASE workflow. CREASE takes as input experimentally measured 1D scattering profiles and identifies as output the key structural features as well as representative 3D real space structures whose computed scattering profiles matches the experimental scattering input.

Figure 1 shows the general workflow in CREASE where experimentally measured 1D scattering profiles are taken as input and CREASE, through an internal optimization, generates as output the key structural features as well as representative 3D real space structures whose computed scattering profiles match the experimental scattering input. If you are interested in this method, you may wish to watch Prof. Jayaraman's recently recorded lecture on CREASE and its uses. The lecture can be found in this link.

In March 2024, the CREASE method was extended and made available as CREASE-2D. This method works directly with experimentally measured 2D scattering profiles and outputs the 3D real space structures which can additionally have structural anisotropy (described further in Section 6).

CHAPTER

TWO

2. GUIDING PHILOSOPHY

CREASE's workflow is based on the **philosophy** that the real-space three-dimensional (amorphous) arrangement of constituents in soft materials can be reduced to a lower dimensional mathematical representation of key 'structural features' and that the distributions of those structural features give rise to a computed scattering profile. *For example, for a system with core-corona spherical micelles at low concentrations, these structural features could be sizes of core and corona and probability distributions of those sizes. At higher concentration, there would be additional structural features that describe the relative neighborhood of each micelle, for example through mathematical order parameters describing positional and orientational order.* The user can decide the types of structural features they are interested in (*e.g.*, any fundamentally interesting structural information and/or structural features that the researcher knows will impact the soft materials' eventual application). Once the users have decided on the key structural features they are interested in, they can use CREASE to run an optimization loop where it iterates over various sets of structural features. In the optimization loop, for each set of structural features CREASE i) calculates the computed scattering profile, I_{comp} (q) (*more about this calculation below*), and ii) compares the I_{comp} (q) profile to the experimental (input) profile, I_{exp} (q), eventually converging towards the sets (*note the intentional use of plural!*) that have I_{comp} (q) profiles most closely matching the input I_{exp} (q).

CHAPTER

THREE

3. CREASE-GA IMPLEMENTATION

CREASE has been implemented in a python code using a simple optimization method - genetic algorithm. CREASE's genetic algorithm (CREASE-GA) takes as input 1D SAXS and/or SANS scattering profile from amorphous soft materials structures. It also requires the user's choice of the types of structural features (i.e., 'genes' of the 'individuals' in GA) based on their knowledge of the general shape of the assembled structure from other imaging techniques and/or subject matter expertise. Then, CREASE-GA starts with an initial 'generation' of multiple sets of structural features (i.e., multiple 'individuals' with specific values of 'genes') and iterates in the GA loop towards the optimal individuals whose genes gives rise to a computed scattering profile, I_{comp} (q), that closely matches the input experimentally measured scattering profile, I_{exp} (q). One important calculation in this loop is the I_{comp} (q) for a given set ('individual') of structural features ('genes'); this has been done so far in one of two ways (**Figure 2**). One way (let us call it *Debye method*) is by creating for each set of genes their representative three-dimensional real space structures filled with point scatterers whose scattering length densities represent the constituents of the system, and using the Debye equation on the scatterer positions to compute I_{comp} (q). This way can be computationally intensive either due to the structure generation step or the Debye calculation despite computational tricks **[1-3]**. Another way is by using a machine learning (ML) model that links the structural features directly to I_{comp} (q); Jayaraman and coworkers have used neural networks trained on thousands of computed scattering profiles calculated from the Debye method for various sets of genes. Using this ML model for I_{comp} (q) calculation can give orders of magnitude speed up over the Debye method, after the initial time investment of training the ML model.

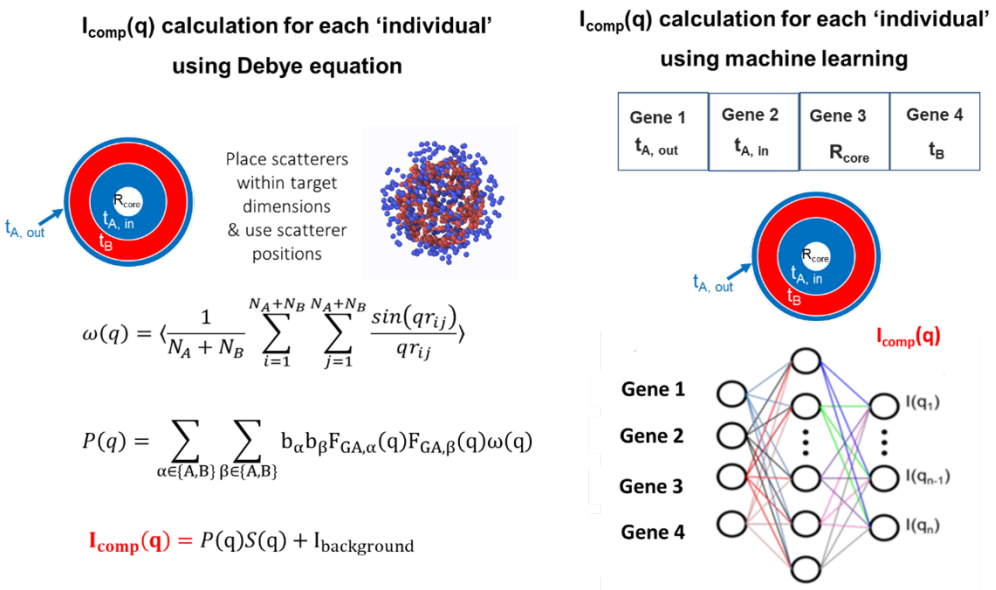

Fig. 1: Figure 2. Two ways to calculate the 'computed scattering profile' or I_{comp} (q) for each 'individual' in the genetic algorithm. Each individual has a unique set of genes (i.e., a set of structural features). Both the Debye equation-based method (left) and machine learning approach (right) connect the values of those structural features or genes to a I_{comp} (q) vs. q profile. In this figure a vesicle with a hollow core, an inner A-layer, a B-layer, and an outer A-layer is used as an example to show that the dimensions related to those layers and core become the 'genes' for this system. There can be additional or fewer genes depending on the user's interest. It is important to note that in the machine learning approach the user may want to represent the structural features (genes) as normalized dimensions to make the trained machine learning model transferrable. (see for example Ref . **[4]**)

CHAPTER

FOUR

4. HOW HAS CREASE BEEN USED SO FAR?

CREASE method has been used to interpret small angle scattering results to

a. *Identify relevant dimensions of assembled structures in polymer solutions at dilute concentrations* **[5-9]**: CREASE has been applied to characterize structure of the 'primary particle' using scattering profiles I(q) ~ P(q) (*i.e.*, conditions where S(q) is ~1) for a variety of 'primary particles' (micelles **[6, 7, 9]**, vesicles **[8]**, and fibrils **[5]**) bypassing the need for an analytical model.

b. *Understand the amorphous structure of spherical particles at high concentrations regardless of extent of mixing/segregation*: CREASE has also been extended to analyze S(q) part of the scattering profiles from concentrated binary mixture of polydisperse spherical nanoparticles (i.e., P(q) is a sphere form factor) to determine the extent of segregation/mixing of the two types of nanoparticles and the precise mixture composition **[4, 10]**.

c. *Elucidate the amorphous structure of particles / micelles in solutions, with unknown primary particle form and unknown assembled/dispersed structure* **[11]**: Most recently, for systems where one does not know the P(q) or S(q) a priori, CREASE has been extended to simultaneously interpret structural information held in P(q) and S(q) and appropriately called 'P(q) and S(q) CREASE' **[11]**.

CREASE has taken as input 1D SAXS profiles and/or SANS profiles: In the studies above, the input to CREASE has been (i) a single SAXS profile of the system, or (ii) one SAXS profile and a one SANS profile of the same system, or (iii) multiple SANS profiles with contrast matching one or the other component(s) in the system with the solvent. Next development steps of CREASE development are focused on 2D profiles for soft materials that show anisotropy in the assembled structure.

CREASE with Debye method vs. ML-model for computed scattering profile calculation: In earlier implementations of CREASE, the Debye method for computed scattering profile calculation was used; as noted above this calculation was initially found to be quite time consuming. In following work, the structure generation (done in every step of Debye method) was found to more computationally intensive while the computed scattering calculations using Debye method have been made faster than in previous implementations. The machine learning (ML) enhanced CREASE-GA, with a well-trained ML model avoids both Debye equation based computed scattering calculation and the three-dimensional real space structure generation in the optimization loop, making is significantly faster than using Debye method (*e.g.*, one can complete CREASE-GA optimization is less than an hour on a laptop with a pre-trained ML model!)

CHAPTER

FIVE

5. UNIQUE ADVANTAGES OF CREASE

Here are some unique advantages of CREASE-GA regardless of availability of appropriate analytical models for the system being characterized:

- The computed scattering profile calculation is done using scatterer placement within structures defined by the 'genes' (i.e., structural features). This treats all soft materials systems in the same way as being composed of scatterers with no detail about the molecules. So, even in the case of polymer chains, there are no chains in this GA step – only scatterers. This overcomes issues one may have not knowing anything about chain conformations (*e.g.*, is Gaussian distribution of chain conformations valid or not?). If one needs information about the chain configurations they can follow up this GA step with an molecular simulation step using models (coarse-grained or atomistic) representing polymers.
- Any structural feature of interest can be a 'gene'; for the same system, two different users may be interested in different structural features. *For example, in the case of vesicles (Figure 2), one user may be interested in all four dimensions (core radius and thickness of every individual layer in the shell leading to four 'genes') and another user may be interested simply in the core radius and shell thickness.* Further, some structural features the user may be interested in may not be in any existing analytical model. **Note:** If and how well CREASE can identify a structural feature reliably from an input experimental scattering profile will depend on how much that structural feature affects the computed scattering profile and how the computed scattering profile changes with the values of the structural features. See for example recent work on analysis of scattering results from methylcellulose fibrils from Wu and Jayaraman **[5]**. In that system, the length, Kuhn length (KL), and diameter of fibrils are the structural features of interest, however Wu and Jayaraman showed that KL values could only be identified well if they were within a certain range of values for the methylcellulose systems. Such sensitivity analysis is very useful in deciding on the genes used in the optimization.
- Genetic algorithm (GA) is the chosen optimization method here because it is easy for others to adopt regardless of prior computational knowledge and experience. Furthermore, GA's output contains multiple individuals whose computed scattering matches experimental scattering. This is useful as it informs us about the degeneracy of solutions for a given experimental profile; in other words, there can be many different structures whose computed scattering profile can match with experiments, so knowing this distribution from the converged 'best match' individuals in the last couple of generations of GA is valuable.
- CREASE also gives as output representative real-space structures – either as is because the system is made of particles whose positions can be generated from scatterer positions or via additional molecular modeling and simulation step to show chain conformations in the structures output from CREASE-GA (*e.g.*, Wessels et al. **[7]**). These structures can then be used as an input for other non-equilibrium simulations or calculations of properties that depend on structure (*e.g.*, resistor network model calculation for electrical conductivity12 and finite-difference time-domain method for optical properties **[13, 14]**).
- One major advantage of Machine learning (ML) enhanced CREASE-GA is the computational speed up. As a result, ML-enhanced CREASE-GA can facilitate high-throughput analysis of related systems as long as the trained ML model, in particular the structural features that are inputs to the ML model, are valid for those related systems.

- CREASE can be used to test the researcher's hypotheses about how the soft materials structures of interest form/evolve with changing conditions. The user is directed to examples of hypothesis testing in the studies presented in Refs. **[5, 11, 15]**

CHAPTER

SIX

6. EXTENSION OF CREASE TO 2D PROFILES: CREASE-2D

As noted in our recent review article **[16]**, the above studies of CREASE worked with the input of 1D SAXS profiles and/or SANS profiles, either (i) a single SAXS profile of the system, or (ii) one SAXS profile and a one SANS profile of the same system, or (iii) multiple SANS profiles with contrast matching one or the other component(s) in the system with the solvent. To extend CREASE to interpret 2D profiles for soft materials that show anisotropy in the assembled structure, Jayaraman and coworkers have now developed CREASE-2D. **[17]** CREASE-2D enables direct interpretation of 2D profile which is far more complex than analysis of 1D scattering profiles, I(q) vs. q, obtained by averaging along all azimuthal angles. Currently, researchers who study materials with any form of anisotropic structure (e.g., processed aligned synthetic conducting fibers, field-driven orientational alignment in polymers for sensing/electronics, sheared formulations during rheological measurements in personal care industry) need to interpret the entire 2D scattering profile. Yet analyses of such 2D profiles have traditionally only been done by fitting analytical models to 1D profiles obtained by averaging along all azimuthal angles or sections of the 2D profile. Such averaging schemes lose key information about the anisotropic structural arrangements that can drive the function of the materials. CREASE-2D method overcomes these current limitations and provides polymer researchers the speed (due to ML surrogate models) and accuracy (by avoiding any averaging of the 2D profile) to interpret quantitative structural information (e.g., domain shapes, sizes, orientation, volume fraction) from the entire 2D scattering profiles without any approximations. The surrogate model used to link structural features to 2D scattering profile was trained on 3D structures generated by a recent developed computational method - Computational Approach for Structure Generation of Anisotropic Particles (CASGAP). **[18]** CASGAP generates representative 3D structures for input desired distribution of particle (representing domain) sizes and shapes and desired spatial orientations without particles overlapping at desired packing density. Using 2400 generated structures generated from CASGAP, Jayaraman and co-workers were able to train the surrogate XG-Boost ML model. Then, using 600 structures (unseen by the surrogate model) they validate the performance of the ML-model as well as the successful performance of the entire CREASE-2D workflow.

6.1 References

1. Brisard, S.; Levitz, P., *Small-angle scattering of dense, polydisperse granular porous media: Computation free of size effects.* **Phys. Rev. E 2013, 87 (1), 013305.** (link)
2. Olds, D. P.; Duxbury, P. M., *Efficient algorithms for calculating small-angle scattering from large model structures.* **Journal of Applied Crystallography 2014, 47 (3), 1077-1086.** (link)
3. Schmidt-Rohr, K., *Simulation of small-angle scattering curves by numerical Fourier transformation.* **Journal of Applied Crystallography 2007, 40 (1), 16-25.** (link)
4. Heil, C. M.; Patil, A.; Dhinojwala, A.; Jayaraman, A., *Computational Reverse-Engineering Analysis for Scattering Experiments (CREASE) with Machine Learning Enhancement to Determine Structure of Nanoparticle Mixtures and Solutions.* **ACS Cent. Sci. 2022, 8 (7), 996-1007.** (link)
5. Wu, Z.; Jayaraman, A., *Machine Learning-Enhanced Computational Reverse-Engineering Analysis for Scattering Experiments (CREASE) for Analyzing Fibrillar Structures in Polymer Solutions.* **Macromolecules 2022, 55 (24), 11076-11091.** (link)

6. Beltran-Villegas, D. J.; Wessels, M. G.; Lee, J. Y.; Song, Y.; Wooley, K. L.; Pochan, D. J.; Jayaraman, A., *Computational Reverse-Engineering Analysis for Scattering Experiments on Amphiphilic Block Polymer Solutions.* **J. Am. Chem. Soc. 2019, 141 (37), 14916-14930.** (link)

7. Wessels, M. G.; Jayaraman, A., *Computational Reverse-Engineering Analysis of Scattering Experiments (CREASE) on Amphiphilic Block Polymer Solutions: Cylindrical and Fibrillar Assembly.* **Macromolecules 2021, 54 (2), 783-796.** (link)

8. Ye, Z.; Wu, Z.; Jayaraman, A., *Computational Reverse Engineering Analysis for Scattering Experiments (CREASE) on Vesicles Assembled from Amphiphilic Macromolecular Solutions.* **JACS Au 2021, 1 (11), 1925-1936.** (link)

9. Wessels, M. G.; Jayaraman, A., *Machine Learning Enhanced Computational Reverse Engineering Analysis for Scattering Experiments (CREASE) to Determine Structures in Amphiphilic Polymer Solutions.* **ACS Polym. Au 2021, 1 (3), 153-164.** (link)

10. Heil, C. M.; Jayaraman, A., *Computational Reverse-Engineering Analysis for Scattering Experiments of Assembled Binary Mixture of Nanoparticles.* **ACS Materials Au 2021, 1 (2), 140.** (link)

11. Heil, C. M.; Ma, Y.; Bharti, B.; Jayaraman, A., *Computational Reverse-Engineering Analysis for Scattering Experiments for Form Factor and Structure Factor Determination ("P(q) and S(q) CREASE").* **JACS Au 2023, 3 (3), 889-904.** (link)

12. White, S. I.; DiDonna, B. A.; Mu, M.; Lubensky, T. C.; Winey, K. I., *Simulations and electrical conductivity of percolated networks of finite rods with various degrees of axial alignment.* **Physical Review B 2009, 79 (2), 024301.** (link)

13. Patil, A.; Heil, C. M.; Vanthournout, B.; Bleuel, M.; Singla, S.; Hu, Z.; Gianneschi, N. C.; Shawkey, M. D.; Sinha, S. K.; Jayaraman, A., *Structural Color Production in Melanin-Based Disordered Colloidal Nanoparticle Assemblies in Spherical Confinement.* **Advanced Optical Materials 2021, 2102162.** (link)

14. Patil, A.; Heil, C. M.; Vanthournout, B.; Singla, S.; Hu, Z.; Ilavsky, J.; Gianneschi, N. C.; Shawkey, M. D.; Sinha, S. K.; Jayaraman, A.; Dhinojwala, A., *Modeling Structural Colors from Disordered One-Component Colloidal Nanoparticle-based Supraballs using Combined Experimental and Simulation Techniques.* **ACS Materials Letters 2022, 4 (9), 1848-1854.** (link)

15. Lee, J. Y.; Song, Y.; Wessels, M. G.; Jayaraman, A.; Wooley, K. L.; Pochan, D. J., *Hierarchical Self-Assembly of Poly(D-glucose carbonate) Amphiphilic Block Copolymers in Mixed Solvents.* **Macromolecules 2020, 53 (19), 8581-8591.** (link)

16. Lu, S.; Jayaraman, A., *Machine Learning for Analyses and Automation of Structural Characterization of Polymer Materials.* **Progress in Polymer Science 2024 (under review).**

17. Akepati, S. V. R.; Gupta, N.; Jayaraman, A., *Computational Reverse Engineering Analysis of the Scattering Experiment Method for Interpretation of 2D Small-Angle Scattering Profiles (CREASE-2D).* **JACS Au 2024, 4, 1570-1582.** (link)

18. Gupta, N.; Jayaraman, A., *Computational Approach for Structure Generation of Anisotropic Particles (CASGAP) with Targeted Distributions of Particle Design and Orientational Order.* **Nanoscale 2023, 15, 14958-14970.** (link)

6.2 Contact us

If you have any questions or feedback, please let us know by emailing creasejayaramanlab AT gmail.com.

6.2.1 Installation

To install this package on a linux or macOS machine, follow these steps:

1. We will be using Anaconda to handle the python environment. First, we need to install anaconda on our machine. You can also consider installing miniconda for faster installation and lower storage consumption *only if you are an advanced user*. We recommend installing the full version of anaconda if you have limited experience with python and/or computer programming.
2. At this point, you will need a bash terminal to work with. All the installation steps *after* this step will need to be accomplished from a terminal, using command line interface.
    - If you are on a linux or MacOS machine

        You can directly launch a terminal.

    - If you are on a Windows machine

        If you have installed the full version of Anaconda/Miniconda in Step 1, the most straightforward way to launch a terminal will be using the *Anaconda prompt* that comes with the conda installation. You should be able to find the *Anaconda prompt* from your Start menu. You can also consider installing Windows Subsytem for Linux (WSL).
3. Download the package.
    - If you have git installed, this can be done using the following command from a terminal:

    ```
    git clone https://github.com/arthijayaraman-lab/crease_ga
    ```

    - You can also directly download the package as a ZIP file from our github webpage by following the guidance here. If you are following this route, you will need to unzip the package to your desired location after the download.
4. Create a new conda environment and install the package along with all the dependencies.
    - *Navigate into the root directory of the cloned package*. If you are using the anaconda prompt, you can look up common windows command line prompts. If you are using a unix-based shell (linux, macOS, or WSL subsystem), you can look up common commands for Unix. Either case, all you will need would probably be *displaying list of files and directories in the current folder* (`dir` for windows, `ls` for unix), and *moving to different directories* (`cd [new_directory_path]` for both windows and unix). You should end up at a directory called `crease_ga` and be able to see files named `setup.py`, `environment.yml` and `README.md` (among others) in the directory.
    - create a fresh conda environment with creaes_ga package and its dependencies installed using

    ```
    conda env create -f environment.yml
    ```

    - Activate the environment with

    ```
    conda activate crease_ga
    ```

To check if the package and its dependencies have been properly installed, you can try running

```
python3
```

to launch python, and then in the resulting python command line, run

```
import crease_ga
crease_ga.__version__
```

If everything is properly installed, you should see the current version number of the package printed. In that case, you are all set to use the `crease_ga` package! Remember to activate the proper environment every time by with `conda activate crease_ga`.

You can run the Jupyter notebook tutorials with the command

```
jupyter notebook
```

- **NOTE**: if you intend to run this on a supercomputing cluster, you will need to follow the steps to create a python environment on the corresponding cluster.

Try the package without installation

- If you would like to first try our package by running our tutorial, you can directly launch a docker image of our environment, and access and interact with our jupyter notebook tutorial from a web browser *without performing any installation steps above*. To do this, click this badge:

6.2.2 Sample Workflow

Crease_ga allows the user to fit scattering profiles only using several lines of python code in an intuitive way. Below is a sample workflow for a user who would like to adopt CREASE to analyze an input scattering profile for vesicles.

```
import crease_ga as cga
#Initialize a model
m = cga.Model(pop_number = 5, generations = 5, nloci = 7)
#load a shape
m.load_shape(shape='vesicle',shape_params=[24,54,0.5,50.4,50.4,0.55,7],
                                  minvalu = (50, 30, 30, 30, 0.1, 0.0, 0.1),
                                  maxvalu = (400, 200, 200, 200, 0.45, 0.45, 4))
#Read Iexp(q) from a file
m.load_iq('../IEXP_DATA/Itot_disper_10_Ain12_B6_Aout12_nLP7_dR0.2.txt')
#Solve
m.solve(output_dir='./test_outputs_1')
```

The user can try this code out by downloading and running the jupyter notebook here.

6.2.3 crease_ga.Model

class crease_ga.**Model**(*pop_number=5*, *generations=10*, *nloci=7*, *yaml_file='x'*)

The basic class that defines the model to be used to solve for a scattering profile.

See also:

crease_ga.adaptaion_params.adaptation_params

Attributes

pop_number: int.
Number of individuals within a generation.

generations: int.
Number of generations to run.

nloci: int.
Number of binary bits to represent each parameter in an individual. The decimal parameter value is converted to binary, with "all 0s" corresponding to the min value and "all 1s" corresponding to the max value. The larger the value, the finer the resultion for each parameter.

adaptation_params: crease_ga.adaptation_params.adaptation_params
Object of adaptation parameters used for this model.

`load_shape`(*shape='vesicle'*, *shape_params=None*, *minvalu=None*, *maxvalu=None*)

Load a shape.

Parameters

shape: str. name of the shape.
Currently supported builtin shapes are "vesicle" and "micelle". Can also specify a shape developed in a crease_ga plugin.

shape_params: list.
Values of shape-specific descriptors. See the API of corresponding shape for details. If not specified, or an incorrect number of shape descriptor values are specified, the default values of the shape-specific descriptors will be loaded.

minvalu,maxvalu: list.
Values of the minimum and maximum boundaries of the parameters to be fit. If not specified, or an incorrect number of input parameter boundaries are specified, the default boundaries of the input parameters of the shape will be loaded.

`solve`(*name='ga_job'*, *verbose=True*, *backend='debye'*, *fitness_metric='log_sse'*, *output_dir='./'*, *n_cores=1*, *converge=10*, *needs_postprocess=False*)

Fit the loaded target I(q) for a set of input parameters that maximize the fitness or minimize the error metric (fitness_metric).

Parameters

name: str.
Title of the current run. A folder of the name will be created under current working directory (output_dir), and all output files will be saved in that folder.

verbose: bool. Default=True.
If verbose is set to True, a figure will be produced at the end of each run, plotting the I(q) resulting from the best individual in the current generation and the target I(q).

Useful for pedagogical purpose on jupyter notebook.

fitness_metric: string. Default='log_sse'.

The metric used to calculate fitness. Currently supported:
"log_sse", sum of squared log10 difference at each q point.

output_dir: string. Default="./"
Path to the working directory.

n_cores: int. Default=1
Number of cores to parallelize over

converge: int. Default=10
The generation countdown to stop optimization. If the minimum error does not improve after converge generations, the optimization ends.

needs_postprocess: bool. Default=False
Some CREASE versions require additional steps. Typically required for S(q) CREASE that outputs a 3D visualization.

6.2.4 crease_ga.shapes.vesicle

class crease_ga.shapes.vesicle.scatterer_generator.**scatterer_generator**(*shape_params=[24, 54, 0.5, 50.4, 50.4, 0.55, 7]*, *minvalu=(50, 30, 30, 30, 0.1, 0.0, 0.1)*, *maxvalu=(400, 200, 200, 200, 0.45, 0.45, 4)*)

The wrapper class for vesicle shape. Default length unit: Angstrom.

See also:

crease_ga.Model.load_shape

Notes

The following 7 shape-specific descriptors are to be specified by user (see *Attributes*) as a list, in the precise order as listed, while calling `Model.load_shape` to load this shape:

num_scatterers:
Number of scatterers used to represent a chain. Default: 24

N:
Number of monomers in a chain. Default: 54

eta_B:
Packing fraction of scatterers in B layer. Default: 0.5

lmono_b:
Diameter of a monomer of chemistry B. Default: 50.4 A

lmono_a:
Diameter of a monomer of chemistry A. Default: 50.4 A

fb:
Fraction of monomers in chain that are of B type. fa = 1-fb. Default: 0.55

nLP:
Number of replicates for each individual. Default: 7

The following 7 parameters are to be predicted, in the precise order as listed, by GA:

R_core:
Core radius. Default [min,max]: [50 A, 400 A]

t_Ain:
Thickness of inner A layer. Default [min,max]: [30 A, 200 A]

t_B:
Thickness of B layer. Default [min,max]: [30 A, 200 A]

t_Aout:
Thickness of outer A layer. Default [min,max]: [30 A, 200 A]

sigma_Ain:
Split of solvophilic scatterers between inner and outer layers. Default [min,max]: [0.1, 0.45]

sigma_R:
Dispersity in vesicle size as implemented in the core radius. Default [min,max]: [0.0, 0.45]

log10(bg):
Negative log10 of background intensity. E.g. an background intensity of 0.001 leads to this value being 3. Default [min,max]: [0.1,4]

`converttoIQ`(*qrange*, *param*)

Calculate computed scattering intensity profile.

Parameters

qrange: numpy.array
q values.

param: numpy.array
Decoded input parameters. See *Notes* section of the class documentation.

Returns

IQid: A numpy array holding I(q).

6.2.5 crease_ga.shapes.micelle

`class crease_ga.shapes.micelle.scatterer_generator.scatterer_generator`(*shape_params=[8, 24, 0.5, 0.5, 50.4, 50.4]*, *minvalu=(2, 0, 0)*, *maxvalu=(60, 1, 5)*)

The wrapper class for micelle shape. Default length unit: Angstrom.

See also:

crease_ga.Model.load_shape

Notes

The following 6 shape-specific descriptors are to be specified by user (see *Attributes*) as a list, in the precise order as listed, while calling `Model.load_shape` to load this shape:

num_scatterers:
Number of scatterers per chain (num_scatterers). Default: 8

N:
Number of beads on chain. Default: 24

fA:
fraction of beads that are of chemistry A. Default: 0.5

rho_core:
Density or volume freaction of the solvophobic block. Default: 0.5

lmono_a:
Monomer contour length (diameter) of chemistry B. Default: 50.4 A

lmono_b:
Monomer contour length (diameter) of chemistry A. Default: 50.4 A

The following 3 parameters are to be predicted, in the precise order as listed, by GA:

N_agg:
Aggregation number. Default [min,max]: [2 A, 60 A]

ecorona:
Fraction of the micelle diameter that is occupied by the corona. Default [min,max]: [0,1]

log10(bg):
Negative log10 of Background intensity. E.g. an background intensity of 0.001 leads to this value being 3. Default [min,max]: [0,5]

`converttoIQ`(*qrange*, *param*)

Calculate computed scattering intensity profile.

Parameters

qrange: numpy.array
q values.

param: numpy.array
Decoded input parameters. See *Notes* section of the class documentation.

Returns

IQid: A numpy array holding I(q).

6.2.6 crease_ga.adaptation_params

`class crease_ga.adaptation_params`(*gdmmin=0.005*, *gdmmax=0.85*, *pcmin=0.1*, *pcmax=1*, *pmmin=0.006*, *pmmax=0.25*, *kgdm=1.1*, *pc=0.6*, *pm=0.001*)

Class for all adaptation parameters needed for a GA run.

Attributes

gdmmin: float. Default=0.005.
The minimum acceptable value of gdm, a measurement of diversity within a generation (high gdm means low diversity, and vice versa). If gdm of the current generation falls below *gdmmin*, *pc* will be multiplied by *kgdm* and *pm* will be divided by *kgdm* to reduce diversity.

gdmmax: float. Default=0.85.
The maximum acceptable value of gdm, a measurement of diversity within a generation (high gdm means low diversity, and vice versa). If gdm of the current generation exceeds *gdmmax*, *pc* will be divided by *kgdm* and *pm* will be multiplied by *kgdm* to increase diversity.

pcmin: float. Default=0.1.
Minimum value of *pc*. *pc* cannot be further adjusted below *pcmin*, even if *gdm* is still too high.

pcmax: float. Default=1.
Maximum value of *pc*. *pc* cannot be further adjusted above *pcmax*, even if *gdm* is still too low.

pmmin: float. Default=0.006.
Minimum value of *pm*. *pm* cannot be further adjusted below *pmmin*, even if *gdm* is still too low.

pmmin: float. Default=0.25.
Maximum value of *pm*. *pm* cannot be further adjusted above *pmmax*, even if *gdm* is still too high.

kgdm: float. Default=1.1.
Should be > 1. The magnitude of adjustment for *pc* and *pm* in case *gdm* falls ouside of [*gdmmin*,`gdmmax`].

pc: float. Default=0.6.
possibility of a crossover action happening on an individual in the next generation. *pc* is updated after each generation according to *gdm*.

pm: float. Default=0.001.
possibillity of a mutation action happening on each gene in an individual. *pm* is updated after each generation according to *gdm*.

`update`(*gdm*)

Update *pc* and *pm* according to a gdm value.

6.2.7 crease_ga.utils

`utils.initial_pop`(*nloci*, *numvars*)

Produce a generation of (binary) chromosomes.

Parameters

popnumber: int
Number of individuals in a population.

nloci: int
Number of binary bits to represent each parameter in a chromosome.

numvars: int
Number of parameters in a chromosome.

Returns

pop: np.array of size (*popnumber*,`nloci`*`numvars`)
A numpy array of binary bits representing the entire generation, with each row representing a chromosome.

`utils.decode`(*indiv*, *nloci*, *minvalu*, *maxvalu*)

Convert a binary chromosome from a generation back to decimal parameter values.

Parameters

pop: np.array.
A numpy array of binary bits representing the entire generation, with each row representing a chromosome.

indiv: int.
The row ID of the chromosome of interest.

nloci: int
Number of binary bits used to represent each parameter in a chromosome.

minvalu, maxvalu: list-like.
The minimum/maximum boundaries (in decimal value) of each parameter. "All-0s" in binary form will be converted to the minimum for a parameter, and "all-1s" will be converted to the maximum.

Returns

param: np.array.
A 1D array containing the decimal values of the input parameters.

6.2.8 Frequently Asked Questions

Do I have to use the Debye scattering equation to evaluate a scattering profile (I(q))? Can I use known analytical models, trained neural network models, etc.?

With the crease_ga package, we aim to provide a general framework for analyzing scattering profiles using a genetic algorithm (GA) as the optimization method.

1. The GA method can be used to analyze and interpret an input scattering profile by assuming a morphology, defining parameters relevant for that morphology (e.g., dimensions, extent of segregation of molecules in various domains within the morphology), placing scatterers within that assumed morphology, calculating the computational scattering profile using the scatterer placements, and finding the values of the parameters whose computed scattering profile closely matches the input scattering profile.
2. The GA method can also be used to optimize the genes by using a machine learning approaches like a pretrained-neural network which avoids the generation of the three-dimensional real space structure and its scattering calculation using the Debye equation, and directly provides a computed scattering profile.
3. The GA method can also be used to fit an analytical model for that assumed morphology; in this case it would be optimizing the parameters defined in the analytical model.

In the above, the first two approaches rely on defining a certain genome (a set of parameters) for the assumed morphology of interest (vesicles, micelles, fibrils, etc.) The detailed implementation of this step and its variations (e.g., incorporation of dispersity, parallelization, etc.) all fall under "shapes".

How does crease_ga support different assembled shapes and morphologies (e.g., vesicle, micelles, fibrils, etc.)?

Currently, we support two built-in "shapes" in the crease_ga codebase:

- the vesicle model as implemented in Ye, Wu and Jayaraman
- the micelle model as implemented in Beltran-Villegas et al.

Both "shapes" are copied as-is from the articles, without any parallelization, to facilitate the usage of these "shapes" in a jupyter notebook for pedagogical purposes. While we plan to expand the codebase to include other "shapes" we have developed and are developing, we also encourage users to implement their own shapes of interest.

6.2.9 Contribute by developing your own plugins

Crease_ga allows users to develop and contribute their own *"shapes"* through external plugins, and crease_ga is equipped to allow external plugins to be discoverable through package metadata. See here for an example plugin (cga_bbvesicle) that implements the analytical model for lipid vesicles in the Brzustowicz and Brunger (2005) paper.

How should I design my plugins?

The plugins should be classes that are set up exactly as one of the builtin shapes. In the source code of the package, a `scatterer_generator` class should exist and contain a `converttoIQ` method that takes as input a list of q-values (`qrange`) and an individual (`param`), and returns a scattering profile I(q) evaluated at these q-values. Other helper methods can be added as you like, but `converttoIQ` is the bare minimum and the only method that will be directly called by crease_ga. It is also advised that for each shape-specific descriptor and input parameter, the definition, default value (for shape-specific parameters), and default min/max bounds (for input parameter) should be explained as docstrings for the `scatterer_generator` class.

What do I need to do to make my plugin discoverable?

You will simply need to specify the entry point for your `scatterer_generator` class in your `setup.py`.

A simplified example of `setup.py` is as following:

```
from setuptools import setup

setup(
    name="name-of-package",
    install_requires="crease_ga",
    entry_points={"crease_ga.plugins":["registered-name-of-plugin=name-of-package.
↪scatterer_generator:scatterer_generator"]},
    py_modules=["name-of-package"],
    )
```

This will allow the plugins to be registered with crease_ga by the name `registered-name-of-plugin`. If both crease_ga and your plugin are installed properly (in the order of crease_ga first, then your plugin), the `scatterer_generator` class of your plugin can be loaded with the following code:

```
import crease_ga.plugins as plugins
a_name_you_prefer = plugins['registered-name-of-plugin'].load()
new_scatterer_generator = a_name_you_prefer()
```

More commonly, instead of explicitly loading the scatterer_generator, all you will need is to load the shape from your plugin to *crease_ga.Model* by using *crease_ga.Model.load_shape(shape='registered-name-of-plugin')*.

6.2.10 Giving us feedback

We would like to hear your feedback about crease_ga. It will be greatly appreciated if you could take some time to fill out this feedback google form after you have gone through the tutorial we prepared. You are also welcome to directly send your suggestions and critiques to creasejayaramanlab AT gmail.com.

INDEX