instagraal Documentation

Release 0.1.4

Lyam Baudry, Hervé Marie-Nelly

Apr 16, 2020

Contents:

1 Reference API **3**
1.1 instagraal 3

2 Indices and tables **25**

Python Module Index **27**

Index **29**

InstaGRAAL is a Hi-C based scaffolder written with large genome assemblies in mind. As such, it also provides a comprehensive framework for generating, storing and analyzing Hi-C data, as well as a number of polishing libraries for post-scaffolding use. The present documentation is dedicated to exposing the various APIs to users who wish to delve deeper into the internal implementation of the scaffolder in order to better understand how it works and possibly extend its functionalities. The software is under continuous development with various incoming overhauls, and the documentation will improve accordingly.

CHAPTER 1

Reference API

1.1 instagraal

1.1.1 instagraal package

Submodules

instagraal.cuda_lib_gl_single module

class instagraal.cuda_lib_gl_single.**sampler**(*use_rippe*, *S_o_A_frags*, *collector_id_repeats*, *frag_dispatcher*, *id_frag_duplicated*, *id_frags_blacklisted*, *n_frags*, *n_new_frags*, *init_n_sub_frags*, *n_new_sub_frags*, *np_rep_sub_frags_id*, *sub_sampled_sparse_matrix*, *np_sub_frags_len_bp*, *np_sub_frags_id*, *np_sub_frags_accu*, *np_sub_frags_2_frags*, *mean_squared_frags_per_bin*, *norm_vect_accu*, *sub_candidates_dup*, *sub_candidates_output_data*, *S_o_A_sub_frags*, *sub_collector_id_repeats*, *sub_frag_dispatcher*, *sparse_matrix*, *mean_value_trans*, *n_iterations*, *is_simu*, *gl_window*, *pos_vbo*, *col_vbo*, *vel*, *pos*, *raw_im_init*, *pbo_im_buffer*, *gl_size_im*)

Bases: object

apply_replay_simu(*id_fA*, *id_fB*, *op_sampled*, *dt*)

approx_all_likelihood_on_zeros()

approx_single_likelihood_on_zeros()

approx_single_likelihood_on_zeros_mut()

approx_single_likelihood_on_zeros_nuisance()

bomb_the_genome()

create_gpu_struct(*data*)

display_current_matrix(*filename*)

dist_inter_genome(*tmp_gpu_vect_frags*)

estimate_parameters(*max_dist_kb*, *size_bin_kb*, *display_graph*)
estimation by least square optimization of Rippe parameters on the experimental data :param max_dist_kb: :param size_bin_kb:

estimate_parameters_rippe(*max_dist_kb*, *size_bin_kb*, *display_graph*)
estimation by least square optimization of Rippe parameters on the experimental data :param max_dist_kb: :param size_bin_kb:

eval_all_sub_likelihood()

eval_likelihood()

eval_likelihood_4_nuisance()

eval_likelihood_init()

eval_likelihood_on_mut(*id_mut*)

explode_genome(*dt*)

extract_current_sub_likelihood()

extract_uniq_mutations(*id_fi*, *id_fj*, *flip_eject*)

f_hic(*x*, *param*)

f_rippe(*x*, *param*)

fill_dist_all_mut()

fill_dist_single()

fill_dist_single_mut(*id_mut*)

free_gpu()

genome_content()

insert_blocks(*id_fA*, *id_fB*, *max_id*)

insert_repeats(*id_f_ins*)

loadProgram(*filename*)

load_gl_cuda_tex_buffer(*im_init*)

load_gl_cuda_vbo()

local_flip(*id_fA*, *mode*, *max_id*)

modify_genome(*n*)

modify_gl_cuda_buffer(*id_fi*, *dt*)

modify_image_thresh(*val*)

modify_param_simu(*param_simu*, *id_val*, *val*)

perform_mutations(*id_fA*, *id_fB*, *max_id*, *is_first*)

pop_out_pop_in(*id_f_pop*, *id_f_ins*, *mode*, *max_id*)

prepare_sparse_call()

return_neighbours(*id_fA*, *delta0*)

return_rippe_vals(*p0*)

set_jumping_distributions_parameters(*delta*)

setup_all_gpu_struct()

setup_distri_frags()

setup_model_parameters(*param*, *d_max*)

setup_rippe_parameters(*param*, *d_max*)

setup_rippe_parameters_4_simu(*kuhn*, *lm*, *slope*, *d*, *val_inter*, *d_max*)

setup_thrust_modules()

show_sub_slice(*id_ctg1*, *id_ctg2*, *id_frag_a*, *id_frag_b*)

slice_sparse_mat(*id_ctg1*, *id_ctg2*, *id_fragA*, *id_fragB*)

sparse_data_2_gpu()

sparse_data_4_gl(*precision*)

step_nuisance_parameters(*dt*, *t*, *n_step*)

step_sampler(*id_frag*, *n_neighbours*, *dt*)

step_sampler_debug(*id_frag*, *n_neighbours*)

temperature(*t*, *n_step*)

test_copy_struct(*id_fA*, *id_f_sampled*, *mode*, *max_id*)

test_thrust()

transloc(*id_fA*, *id_fB*, *max_id*)

update_neighbourhood()

instagraal.fragment module

class instagraal.fragment.**basic_fragment**

Bases: object

classmethod initiate(*np_id_abs*, *id_init*, *init_contig*, *curr_id*, *start_pos*, *end_pos*, *length_kb*, *gc_content*, *init_frag_start*, *init_frag_end*, *sub_frag_start*, *sub_frag_end*, *super_index*, *id_contig*, *n_accu_frags*)

class instagraal.fragment.**fragment**

Bases: object

classmethod copy(*frag*)

classmethod initiate(*np_id_abs*, *id_init*, *init_contig*, *curr_id*, *start_pos*, *end_pos*, *length_kb*, *gc_content*)

update_name(*contig_id*)

instagraal.glutil module

instagraal.glutil.**draw_axes**()

instagraal.glutil.**draw_line**(*v1*, *v2*)

instagraal.glutil.**init**(*width*, *height*)

instagraal.glutil.**lights**()

instagraal.gpustruct module

class instagraal.gpustruct.**GPUStruct**(*objs*)

Bases: object

copy_from_gpu(*skip=None*)

copy_to_gpu(*skip=None*)

get_packed()

get_ptr()

instagraal.init_nuisance module

instagraal.init_nuisance.**estimate_max_dist_intra**(*p*, *val_inter*)

instagraal.init_nuisance.**estimate_param_hic**(*y_meas*, *x_bins*)

instagraal.init_nuisance.**log_residuals**(*param*, *y*, *x*)

instagraal.init_nuisance.**log_residuals_4_min**(*param*, *y*, *x*)

instagraal.init_nuisance.**peval**(*x*, *param*)

instagraal.init_nuisance.**residual_4_max_dist**(*x*, *p*)

instagraal.init_nuisance.**residuals**(*param*, *y*, *x*)

instagraal.instagraal module

Large genome reassembly based on Hi-C data.

Usage:

instagraal <hic_folder> <reference.fa> [<output_folder>] [–level=4] [–cycles=100] [–coverage-std=1] [–neighborhood=5] [–device=0] [–circular] [–bomb] [–save-matrix] [–pyramid-only] [–save-pickle] [–simple] [–quiet] [–debug]

Options:

-h, --help	Display this help message.
--version	Display the program's current version.

-l 4, –level 4 Level (resolution) of the contact map. Increasing level by one means a threefold smaller resolution but also a threefold faster computation time. [default: 4]

-n 100, –cycles 100 Number of iterations to perform for each bin. (row/column of the contact map). A high number of cycles has diminishing returns but there is a necessary minimum for assembly convergence. [default: 100]

-c 1, –coverage-std 1 Number of standard deviations below the mean. coverage, below which fragments should be filtered out prior to binning. [default: 1]

-N 5, –neighborhood 5 Number of neighbors to sample for potential mutations for each bin. [default: 5]

–device 0 If multiple graphic cards are available, select a specific device (numbered from 0). [default: 0]

-C, --circular	Indicates genome is circular. [default: False]
-b, --bomb	Explode the genome prior to scaffolding. [default: False]
--pyramid-only	Only build multi-resolution contact maps (pyramids) and don't do any scaffolding. [default: False]
--save-pickle	Dump all info from the instaGRAAL run into a pickle. Primarily for development purposes, but also for advanced post hoc introspection. [default: False]
--save-matrix	Saves a preview of the contact map after each cycle. [default: False]
--simple	Only perform operations at the edge of the contigs. [default: False]
--quiet	Only display warnings and errors as outputs. [default: False]
--debug	Display debug information. For development purposes only. Mutually exclusive with –quiet, and will override it. [default: False]

instagraal.instagraal.**main**()

class instagraal.instagraal.**window**(*name*, *folder_path*, *fasta*, *device*, *level*, *n_iterations_em*, *n_iterations_mcmc*, *is_simu*, *scrambled*, *perform_em*, *use_rippe*, *gl_size_im*, *sample_param*, *thresh_factor*, *output_folder*)

Bases: object

A window displaying the live movie of the calculations performed by the scaffolder.

[description]

Parameters

- **name** (*str*) – The name of the project. Will determine the window title.
- **folder_path** (*str or pathlib.Path*) – The directory containing the Hi-C conact map.
- **fasta** (*str or pathlib.Path*) – The path to the reference genome in FASTA format.
- **device** (*int*) – The identifier of the graphic card to be used, numbered from 0. If only one is available, it should be 0.
- **level** (*int*) – The level (resolution) at which to perform scaffolding.
- **n_iterations_em** (*int*) – The number of EM (expectation maximization) iterations.
- **n_iterations_mcmc** (*int*) – The number of MCMC (Markov chain Monte-Carlo) iterations.
- **is_simu** (*bool*) – Whether the parameters should be simulated. Mutually exclusive with use_rippe and will override it.
- **scrambled** (*bool*) – Whether to scramble the genome.
- **perform_em** (*bool*) – Whether to perform EM (expectation maximization).
- **use_rippe** (*bool*) – Whether to explicitly use the model from Rippe et al., 2001.

- **gl_size_im** (*int*) – The size of the window to be displayed.
- **sample_param** (*bool*) – Whether to sample the parameters.
- **thresh_factor** (*float*) – The sparsity (coverage) threshold below which fragments are discarded, as a number of standard deviations below the mean.
- **output_folder** (*str or pathlib.Path*) – The path to the output folder where the scaffolded genome and other relevant information will be saved.

cuda_gl_init()

debug_test_EM(*delta*)

debug_test_model(*id_fi*, *delta*)

draw()
Render the particles

full_em(*n_cycles*, *n_neighbours*, *bomb*, *id_start_sample_param*, *save_matrix=False*)

glinit()

glut_print(*x*, *y*, *font*, *text*, *r*, *g*, *b*, *a*)

modify_image_thresh(*val*)
modify threshold of the matrix

on_click(*button*, *state*, *x*, *y*)

on_key(**args*)

on_mouse_motion(*x*, *y*)

remote_update()

render()

render_image()

replay_simu(*file_simu*, *file_likelihood*, *file_n_contigs*, *file_distances*)

save_behaviour_to_txt()

setup_simu(*id_f_ins*)

simple_start(*n_cycles*, *n_neighbours*, *bomb*)

start_EM()

start_EM_all()

start_EM_no_scrambled()

start_EM_nuisance()

start_MCMC()

start_MTM()

test_model(*id_fi*, *delta*)

timer(*t*)

instagraal.leastsqbound module

Constrained multivariate Levenberg-Marquardt optimization

An updated version of this file can be found at https://github.com/jjhelmus/leastsqbound-scipy

The version here has known bugs which have been fixed above, proceed at your own risk.

- Jonathan J. Helmus (jjhelmus@gmail.com)

`instagraal.leastsqbound.`**`calc_cov_x`**(*infodic*, *p*)
Calculate cov_x from fjac, ipvt and p as is done in leastsq

`instagraal.leastsqbound.`**`err`**(*p*, *bounds*, *efunc*, *args*)

`instagraal.leastsqbound.`**`external2internal`**(*xe*, *bounds*)
Convert a series of external variables to internal variables

`instagraal.leastsqbound.`**`i2e_cov_x`**(*xi*, *bounds*, *cov_x*)

`instagraal.leastsqbound.`**`internal2external`**(*xi*, *bounds*)
Convert a series of internal variables to external variables

`instagraal.leastsqbound.`**`internal2external_grad`**(*xi*, *bounds*)
Calculate the internal to external gradiant

Calculates the partial of external over internal

`instagraal.leastsqbound.`**`leastsqbound`**(*func*, *x0*, *bounds*, *args=()*, ***kw*)
Constrained multivariant Levenberg-Marquard optimization

Minimize the sum of squares of a given function using the Levenberg-Marquard algorithm. Contraints on parameters are inforced using variable transformations as described in the MINUIT User's Guide by Fred James and Matthias Winkler.

Parameters:

- func functions to call for optimization.
- x0 Starting estimate for the minimization.
- **bounds (min,max) pair for each element of x, defining the bounds on** that parameter. Use None for one of min or max when there is no bound in that direction.
- args Any extra arguments to func are places in this tuple.

Returns: (x,{cov_x,infodict,mesg},ier)

Return is described in the scipy.optimize.leastsq function. x and con_v are corrected to take into account the parameter transformation, infodic is not corrected.

Additional keyword arguments are passed directly to the scipy.optimize.leastsq algorithm.

instagraal.linkage module

Genetic map based validation

Aggregate genetic linkage data with an existing assembly for correction and polishing purposes.

Usage:

linkage.py <linkage.csv> <info_frags.txt> [–output <output_file>] [–fasta <reference.fa>]

Options:

-h, --help	Display this help message.

--version	Display the program's current version.
-o, --output	Prefix of output files
-f, --fasta	If a fasta reference is supplied, also write the corresponding genome from info_frags.txt

instagraal.linkage.**collapse_degenerate_markers**(*linkage_records*)

Group all markers with no genetic distance as distinct features to generate a BED file with.

Simple example with sixteen degenerate markers:

```
>>> marker_features = [
... ['36915_sctg_207_31842', 1, 0, 207, 31842],
... ['36941_sctg_207_61615', 1, 0, 207, 61615],
... ['36956_sctg_207_77757', 1, 0, 207, 77757],
... ['36957_sctg_207_78332', 1, 0, 207, 78332],
... ['36972_sctg_207_94039', 1, 0, 207, 94039],
... ['36788_sctg_207_116303', 1, 0.652, 207, 116303],
... ['36812_sctg_207_158925', 1, 1.25, 207, 158925],
... ['36819_sctg_207_165424', 1, 1.25, 207, 165424],
... ['36828_sctg_207_190813', 1, 1.25, 207, 190813],
... ['36830_sctg_207_191645', 1, 1.25, 207, 191645],
... ['36834_sctg_207_195961', 1, 1.25, 207, 195961],
... ['36855_sctg_207_233632', 1, 1.25, 207, 233632],
... ['36881_sctg_207_258658', 1, 1.25, 207, 258658],
... ['82072_sctg_486_41893', 1, 3.756, 486, 41893],
... ['85634_sctg_516_36614', 1, 3.756, 516, 36614],
... ['85638_sctg_516_50582', 1, 3.756, 516, 50582]]
```

```
>>> len(marker_features)
16
```

```
>>> collapsed_features = collapse_degenerate_markers(marker_features)
>>> len(collapsed_features)
5
```

The degenerate features (identical linkage group, genetic distance and original scaffold) are collapsed into a region:

```
>>> collapsed_features[0]
[1, 31842, 94039, 207]
```

The format is [linkage group, start, end, original scaffold].

If a singleton (non-degenerate) feature is found, the region is simply a single point in the genome:

```
>>> collapsed_features[1]
[1, 116303, 116303, 207]
```

so 'start' and 'end' are identical.

Two markers are not considered degenerate if they belong to different original scaffolds, even if they are in terms of genetic linkage:

```
>>> collapsed_features[2]
[1, 158925, 258658, 207]
>>> collapsed_features[3:]
[[1, 41893, 41893, 486], [1, 36614, 50582, 516]]
```

instagraal.linkage.**compare_orderings**(*info_frags_records*, *linkage_orderings*)

Given linkage groups and info_frags records, link pseudo-chromosomes to scaffolds based on the initial contig composition of each group. Because info_frags records are usually richer and may contain contigs not found in linkage groups, those extra sequences are discarded.

Example with two linkage groups and two chromosomes:

```
>>> linkage_orderings = {
...     'linkage_group_1': [
...         ['sctg_516', -3, 36614, 50582, 1],
...         ['sctg_486', -3, 41893, 41893, 1],
...         ['sctg_486', -3, 50054, 62841, 1],
...         ['sctg_207', -3, 31842, 94039, 1],
...         ['sctg_558', -3, 51212, 54212, 1],
...     ],
...     'linkage_group_2': [
...         ['sctg_308', -3, 15892, 25865, 1],
...         ['sctg_842', -3, 0, 8974, 1],
...         ['sctg_994', -3, 0, 81213, 1],
...     ],
... }
>>> info_frags = {
...     'scaffold_A': [
...         ['sctg_308', 996, 15892, 25865, 1],
...         ['sctg_778', 1210, 45040, 78112, -1],
...         ['sctg_842', 124, 0, 8974, 1],
...     ],
...     'scaffold_B': [
...         ['sctg_516', 5, 0, 38000, 1],
...         ['sctg_486', 47, 42050, 49000, 1],
...         ['sctg_1755', 878, 95001, 10844, -1],
...         ['sctg_842', 126, 19000, 26084, 1],
...         ['sctg_207', 705, 45500, 87056, 1],
...     ],
...     'scaffold_C': [
...        ['sctg_558', 745, 50045, 67851, 1],
...        ['sctg_994', 12, 74201, 86010, -1],
...     ],
... }
>>> matching_pairs = compare_orderings(info_frags, linkage_orderings)
>>> matching_pairs['scaffold_B']
(3, 'linkage_group_1', {'sctg_558': 'sctg_207'})
>>> matching_pairs['scaffold_A']
(2, 'linkage_group_2', {'sctg_994': 'sctg_842'})
```

instagraal.linkage.**get_missing_blocks**(*info_frags_records*, *matching_pairs*, *linkage_orderings*)

Get missing blocks in a scaffold based on the genetic map order.

Given matching scaffold blocks/genetic map blocks (based on restriction sites and SNP markers, respectively), move around the scaffold blocks such that they map the genetic map order.

Parameters

- **info_frags_records** (*dict*) – A dictionary representing the scaffolds and their block order as described in an info_frags.txt file
- **matching_pairs** (*dict*) – A list of best matching pairs in the form (scaffold_block, linkage group)

- **linkage_orderings** (*dict*) – A dictionary representing the genetic map and the linkage groups as described in csv file.

Example

```
>>> linkage_orderings = {
...     'linkage_group_1': [
...         ['sctg_516', -3, 36614, 50582, 1],
...         ['sctg_486', -3, 41893, 41893, 1],
...         ['sctg_486', -3, 50054, 62841, 1],
...         ['sctg_207', -3, 31842, 94039, 1],
...         ['sctg_558', -3, 51212, 54212, 1],
...     ],
...     'linkage_group_2': [
...         ['sctg_308', -3, 15892, 25865, 1],
...         ['sctg_842', -3, 0, 8974, 1],
...         ['sctg_994', -3, 0, 81213, 1],
...     ],
... }
>>> info_frags = {
...     'scaffold_A': [
...         ['sctg_308', 996, 15892, 25865, 1],
...         ['sctg_778', 1210, 45040, 78112, -1],
...         ['sctg_842', 124, 0, 8974, 1],
...     ],
...     'scaffold_B': [
...         ['sctg_516', 5, 0, 38000, 1],
...         ['sctg_486', 47, 42050, 49000, 1],
...         ['sctg_1755', 878, 95001, 10844, -1],
...         ['sctg_842', 126, 19000, 26084, 1],
...         ['sctg_207', 705, 45500, 87056, 1],
...     ],
...     'scaffold_C': [
...        ['sctg_558', 745, 50045, 67851, 1],
...        ['sctg_994', 12, 74201, 86010, -1],
...     ],
... }
>>> matching_pairs = compare_orderings(info_frags, linkage_orderings)
>>> new_records = get_missing_blocks(info_frags, matching_pairs,
...                                  linkage_orderings)
>>> for my_bin in new_records['scaffold_A']:
...     print(list(my_bin))
...
['sctg_308', 996, 15892, 25865, 1]
['sctg_778', 1210, 45040, 78112, -1]
['sctg_842', 124, 0, 8974, 1]
['sctg_842', 126, 19000, 26084, 1]
['sctg_994', 12, 74201, 86010, -1]
```

```
>>> for my_bin in new_records['scaffold_B']:
...     print(list(my_bin))
...
['sctg_516', 5, 0, 38000, 1]
['sctg_486', 47, 42050, 49000, 1]
['sctg_1755', 878, 95001, 10844, -1]
['sctg_207', 705, 45500, 87056, 1]
```

(continues on next page)

(continued from previous page)

```
['sctg_558', 745, 50045, 67851, 1]
```

instagraal.linkage.**linkage_group_ordering**(*linkage_records*)

Convert degenerate linkage records into ordered info_frags-like records for comparison purposes.

Simple example:

```
>>> linkage_records = [
... ['linkage_group_1', 31842, 94039, 'sctg_207'],
... ['linkage_group_1', 95303, 95303, 'sctg_207'],
... ['linkage_group_2', 15892, 25865, 'sctg_308'],
... ['linkage_group_2', 41893, 41893, 'sctg_486'],
... ['linkage_group_3', 36614, 50582, 'sctg_516'],
... ]
>>> ordering = linkage_group_ordering(linkage_records)
```

Each key of the record is a newly-formed 'scaffold' (linkage group):

```
>>> sorted(ordering.keys())
['linkage_group_1', 'linkage_group_2', 'linkage_group_3']
```

Records are in the form [init_contig, frag_id, start, end, orientation]. Since fragment ids are meaningless in non-HiC contexts a negative identifier is set so it is understood that region was added due to linkage data (-1 is for recovering data after first-pass polishing and -2 is for sequence insertions after long read based polishing).

```
>>> ordering['linkage_group_1']
[['sctg_207', -3, 31842, 94039, 1], ['sctg_207', -3, 95303, 95303, 1]]
>>> ordering['linkage_group_2']
[['sctg_308', -3, 15892, 25865, 1], ['sctg_486', -3, 41893, 41893, 1]]
```

Orientations are always set to 1 by default.

```
>>> ordering['linkage_group_3']
[['sctg_516', -3, 36614, 50582, 1]]
```

instagraal.linkage.**main**()

instagraal.linkage.**parse_linkage_csv**(*fname*, ***, *dtype=None*, *comments='#'*, *delimiter=None*, *skip_header=1*, *skip_footer=0*, *converters=None*, *missing_values=None*, *filling_values=None*, *usecols=None*, *names=None*, *excludelist=None*, *deletechars=" !#$%&'()*+, -./:;<=>?@[\\]^{|}~"*, *replace_space='_'*, *autostrip=False*, *case_sensitive=True*, *defaultfmt='f%i'*, *unpack=None*, *usemask=False*, *loose=True*, *invalid_raise=True*, *max_rows=None*, *encoding='utf-8'*)

Load data from a text file, with missing values handled as specified.

Each line past the first *skip_header* lines is split at the *delimiter* character, and characters following the *comments* character are discarded.

Parameters

- **fname** (*file, str, pathlib.Path, list of str, generator*) – File, filename, list, or generator to read. If the filename extension is *.gz* or *.bz2*, the file is first decompressed. Note that generators must return byte strings. The strings in a list or produced by a generator are treated as lines.

- **dtype** (*dtype, optional*) – Data type of the resulting array. If None, the dtypes will be determined by the contents of each column, individually.
- **comments** (*str, optional*) – The character used to indicate the start of a comment. All the characters occurring on a line after a comment are discarded
- **delimiter** (*str, int, or sequence, optional*) – The string used to separate values. By default, any consecutive whitespaces act as delimiter. An integer or sequence of integers can also be provided as width(s) of each field.
- **skiprows** (*int, optional*) – *skiprows* was removed in numpy 1.10. Please use *skip_header* instead.
- **skip_header** (*int, optional*) – The number of lines to skip at the beginning of the file.
- **skip_footer** (*int, optional*) – The number of lines to skip at the end of the file.
- **converters** (*variable, optional*) – The set of functions that convert the data of a column to a value. The converters can also be used to provide a default value for missing data: `converters = {3: lambda s: float(s or 0)}`.
- **missing** (*variable, optional*) – *missing* was removed in numpy 1.10. Please use *missing_values* instead.
- **missing_values** (*variable, optional*) – The set of strings corresponding to missing data.
- **filling_values** (*variable, optional*) – The set of values to be used as default when the data are missing.
- **usecols** (*sequence, optional*) – Which columns to read, with 0 being the first. For example, `usecols = (1, 4, 5)` will extract the 2nd, 5th and 6th columns.
- **names** (*{None, True, str, sequence}, optional*) – If *names* is True, the field names are read from the first line after the first *skip_header* lines. This line can optionally be proceeded by a comment delimiter. If *names* is a sequence or a single-string of comma-separated names, the names will be used to define the field names in a structured dtype. If *names* is None, the names of the dtype fields will be used, if any.
- **excludelist** (*sequence, optional*) – A list of names to exclude. This list is appended to the default list ['return','file','print']. Excluded names are appended an underscore: for example, *file* would become *file_*.
- **deletechars** (*str, optional*) – A string combining invalid characters that must be deleted from the names.
- **defaultfmt** (*str, optional*) – A format used to define default field names, such as "f%i" or "f_%02i".
- **autostrip** (*bool, optional*) – Whether to automatically strip white spaces from the variables.
- **replace_space** (*char, optional*) – Character(s) used in replacement of white spaces in the variables names. By default, use a '_'.
- **case_sensitive** (*{True, False, 'upper', 'lower'}, optional*) – If True, field names are case sensitive. If False or 'upper', field names are converted to upper case. If 'lower', field names are converted to lower case.
- **unpack** (*bool, optional*) – If True, the returned array is transposed, so that arguments may be unpacked using `x, y, z = loadtxt(...)`

- **usemask** (*bool, optional*) – If True, return a masked array. If False, return a regular array.
- **loose** (*bool, optional*) – If True, do not raise errors for invalid values.
- **invalid_raise** (*bool, optional*) – If True, an exception is raised if an inconsistency is detected in the number of columns. If False, a warning is emitted and the offending lines are skipped.
- **max_rows** (*int, optional*) – The maximum number of rows to read. Must not be used with skip_footer at the same time. If given, the value must be at least 1. Default is to read the entire file.

  New in version 1.10.0.
- **encoding** (*str, optional*) – Encoding used to decode the inputfile. Does not apply when *fname* is a file object. The special value 'bytes' enables backward compatibility workarounds that ensure that you receive byte arrays when possible and passes latin1 encoded strings to converters. Override this value to receive unicode arrays and pass strings as input to converters. If set to None the system default is used. The default value is 'bytes'.

  New in version 1.14.0.

Returns **out** – Data read from the text file. If *usemask* is True, this is a masked array.

Return type ndarray

See also:

numpy.loadtxt() equivalent function when no data is missing.

Notes

- When spaces are used as delimiters, or when no delimiter has been given as input, there should not be any missing data between two fields.
- When the variables are named (either by a flexible dtype or with *names*, there must not be any header in the file (else a ValueError exception is raised).
- Individual values are not stripped of spaces by default. When using a custom converter, make sure the function does remove spaces.

References

Examples

```
>>> from io import StringIO
>>> import numpy as np
```

Comma delimited file with mixed dtype

```
>>> s = StringIO(u"1,1.3,abcde")
>>> data = np.genfromtxt(s, dtype=[('myint','i8'),('myfloat','f8'),
... ('mystring','S5')], delimiter=",")
>>> data
array((1, 1.3, b'abcde'),
      dtype=[('myint', '<i8'), ('myfloat', '<f8'), ('mystring', 'S5')])
```

Using dtype = None

```
>>> _ = s.seek(0) # needed for StringIO example only
>>> data = np.genfromtxt(s, dtype=None,
... names = ['myint','myfloat','mystring'], delimiter=",")
>>> data
array((1, 1.3, b'abcde'),
      dtype=[('myint', '<i8'), ('myfloat', '<f8'), ('mystring', 'S5')])
```

Specifying dtype and names

```
>>> _ = s.seek(0)
>>> data = np.genfromtxt(s, dtype="i8,f8,S5",
... names=['myint','myfloat','mystring'], delimiter=",")
>>> data
array((1, 1.3, b'abcde'),
      dtype=[('myint', '<i8'), ('myfloat', '<f8'), ('mystring', 'S5')])
```

An example with fixed-width columns

```
>>> s = StringIO(u"11.3abcde")
>>> data = np.genfromtxt(s, dtype=None, names=['intvar','fltvar','strvar'],
...     delimiter=[1,3,5])
>>> data
array((1, 1.3, b'abcde'),
      dtype=[('intvar', '<i8'), ('fltvar', '<f8'), ('strvar', 'S5')])
```

An example to show comments

```
>>> f = StringIO('''
... text,# of chars
... hello world,11
... numpy,5''')
>>> np.genfromtxt(f, dtype='S12,S12', delimiter=',')
array([(b'text', b''), (b'hello world', b'11'), (b'numpy', b'5')],
  dtype=[('f0', 'S12'), ('f1', 'S12')])
```

instagraal.linkage.**write_bed**(*records*, *output_file*)

instagraal.log module

Basic logging setup for instaGRAAL.

Logging level can be set by the user and determines the verbosity of the whole program.

instagraal.optim_rippe_curve_update module

instagraal.optim_rippe_curve_update.**estimate_max_dist_intra**(*p*, *val_inter*)

instagraal.optim_rippe_curve_update.**estimate_max_dist_intra_nuis**(*p*, *val_inter*, *old_s*)

instagraal.optim_rippe_curve_update.**estimate_param_rippe**(*y_meas*, *x_bins*)

instagraal.optim_rippe_curve_update.**log_peval**(*x*, *param*)

instagraal.optim_rippe_curve_update.**log_residuals**(*p*, *y*, *x*)

instagraal.optim_rippe_curve_update.**peval**(*x*, *param*)

instagraal.optim_rippe_curve_update.**residual_4_max_dist**(*x*, *p*)

instagraal.optim_rippe_curve_update.**residuals**(*p*, *y*, *x*)

instagraal.parse_info_frags module

A couple of functions to manipulate info_frags.txt files and BED files.

This module started as a couple scripts to flip back artefact inversions introduced by the GRAAL assembler and its derivatives (see: https://github.com/koszullab/GRAAL, https://github.com/koszullab/instaGRAAL). It has greatly evolved since, and provides a range of functions to polish assemblies and correct potential missassemblies.

instagraal.parse_info_frags.**correct_scaffolds**(*scaffolds*, *corrector*)

Unfinished

instagraal.parse_info_frags.**correct_spurious_inversions**(*scaffolds*, *criterion='colinear'*)

Invert bins based on orientation neighborhoods. Neighborhoods can be defined by three criteria:

-a 'cis' neighborhood is a group of bins belonging to the same initial contig -a 'colinear' neighborhood is a 'cis' neighborhood where bins are ordered the same way they were on the initial contig -a 'contiguous' neighborhood is a 'colinear' neighborhood where all bins are exactly consecutive, i.e. the end position of each bin matches the starting position of the next bin

This function looks for such neighborhoods and orients all bins in it according to the majority orientation.

An example with three inversions, one for each criterion:

```
>>> scaffolds = {
...     "scaffold1": [
...         ["contig1", 1, 100, 200, 1],
...         ["contig1", 2, 200, 300, 1],
...         ["contig1", 3, 300, 400, -1], # <-- inversion (contiguous)
...         ["contig1", 4, 400, 500, 1],
...         ["contig1", 10, 1500, 1605, 1],
...         ["contig1", 12, 1750, 1850, -1], # <-- inversion (colinear)
...         ["contig1", 23, 2100, 2499, 1],
...         ["contig1", 28, 2850, 3000, 1],
...         ["contig1", 0, 0, 100, -1], # <-- inversion (cis)
...         ["contig2", 554, 1850, 1900, -1],
...     ],
... }
```

With the 'cis' criterion, pretty much all bins from "contig1" get inverted to the majority orientation (+):

```
>>> sc_cis = correct_spurious_inversions(scaffolds, "cis")
>>> for my_bin in sc_cis['scaffold1']:
...     print(my_bin)
...
['contig1', 1, 100, 200, 1]
['contig1', 2, 200, 300, 1]
['contig1', 3, 300, 400, 1]
['contig1', 4, 400, 500, 1]
['contig1', 10, 1500, 1605, 1]
['contig1', 12, 1750, 1850, 1]
['contig1', 23, 2100, 2499, 1]
['contig1', 28, 2850, 3000, 1]
['contig1', 0, 0, 100, 1]
['contig2', 554, 1850, 1900, -1]
```

With the 'colinear' criterion, the bin ['contig1', 0, 0, 100, -1] is treated as a different neighborhood from the rest (as it is not colinear with the other bins from 'contig1') and remains untouched:

```
>>> sc_colinear = correct_spurious_inversions(scaffolds, "colinear")
>>> for my_bin in sc_colinear['scaffold1']:
...     print(my_bin)
...
['contig1', 1, 100, 200, 1]
['contig1', 2, 200, 300, 1]
['contig1', 3, 300, 400, 1]
['contig1', 4, 400, 500, 1]
['contig1', 10, 1500, 1605, 1]
['contig1', 12, 1750, 1850, 1]
['contig1', 23, 2100, 2499, 1]
['contig1', 28, 2850, 3000, 1]
['contig1', 0, 0, 100, -1]
['contig2', 554, 1850, 1900, -1]
```

With the 'contiguous' criterion, the ['contig1', 12, 1750, 1850, -1] breaks with the contiguous region spanning from 100 to 400 bp on 'contig1' and so is treated as a different neighborhood as well:

```
>>> sc_cont = correct_spurious_inversions(scaffolds, "contiguous")
>>> for my_bin in sc_cont['scaffold1']:
...     print(my_bin)
...
['contig1', 1, 100, 200, 1]
['contig1', 2, 200, 300, 1]
['contig1', 3, 300, 400, 1]
['contig1', 4, 400, 500, 1]
['contig1', 10, 1500, 1605, 1]
['contig1', 12, 1750, 1850, -1]
['contig1', 23, 2100, 2499, 1]
['contig1', 28, 2850, 3000, 1]
['contig1', 0, 0, 100, -1]
['contig2', 554, 1850, 1900, -1]
```

Note that 'contig2' remains untouched at all times since bins in it are never in the same neighborhood as those from 'contig1'.

instagraal.parse_info_frags.**find_lost_dna**(*init_fasta*, *scaffolds*, *output_file=None*)

instagraal.parse_info_frags.**format_info_frags**(*info_frags*)
A function to seamlessly run on either scaffold dictionaries or info_frags.txt files without having to check the input first.

instagraal.parse_info_frags.**integrate_lost_dna**(*scaffolds*, *lost_dna_positions*)

instagraal.parse_info_frags.**is_block**(*bin_list*)
Check if a bin list has exclusively consecutive bin ids.

instagraal.parse_info_frags.**main**()

instagraal.parse_info_frags.**parse_bed**(*bed_file*)
Import a BED file (where the data entries are analogous to what may be expected in an info_frags.txt file) and return a scaffold dictionary, similarly to parse_info_frags.

instagraal.parse_info_frags.**parse_info_frags**(*info_frags*)
Import an info_frags.txt file and return a dictionary where each key is a newly formed scaffold and each value is the list of bins and their origin on the initial scaffolding.

instagraal.parse_info_frags.**plot_info_frags**(*scaffolds*)
A crude way to visualize new scaffolds according to their origin on the initial scaffolding. Each scaffold spawns a new plot. Orientations are represented by different colors.

instagraal.parse_info_frags.**rearrange_intra_scaffolds**(*scaffolds*)
Rearranges all bins within each scaffold such that all bins belonging to the same initial contig are grouped together in the same order. When two such groups are found, the smaller one is moved to the larger one.

instagraal.parse_info_frags.**remove_spurious_insertions**(*scaffolds*)
Remove all bins whose left and right neighbors belong to the same, different scaffold.

Example with three such insertions in two different scaffolds:

```
>>> scaffolds = {
...     "scaffold1": [
...         ["contig1", 0, 0, 100, 1],
...         ["contig1", 1, 100, 200, 1],
...         ["contig23", 53, 1845, 2058, -1], # <-- insertion
...         ["contig1", 4, 254, 408, 1],
...         ["contig1", 7, 805, 1253, 1],
...         ["contig5", 23, 1500, 1605, -1],
...         ["contig65", 405, 32145, 45548, -1], # <-- insertion
...         ["contig5", 22, 1385, 1499, -1],
...     ],
...     "scaffold2": [
...         ["contig8", 0, 0, 250, 1],
...         ["contig17", 2454, 8754, -1], # <-- insertion
...         ["contig8", 2, 320, 480, 1],
...      ],
... }
```

```
>>> new_scaffolds = remove_spurious_insertions(scaffolds)
>>> for my_bin in new_scaffolds['scaffold1']:
...     print(my_bin)
...
['contig1', 0, 0, 100, 1]
['contig1', 1, 100, 200, 1]
['contig1', 4, 254, 408, 1]
['contig1', 7, 805, 1253, 1]
['contig5', 23, 1500, 1605, -1]
['contig5', 22, 1385, 1499, -1]
```

```
>>> for my_bin in new_scaffolds['scaffold2']:
...     print(my_bin)
...
['contig8', 0, 0, 250, 1]
['contig8', 2, 320, 480, 1]
```

instagraal.parse_info_frags.**reorient_consecutive_blocks**(*scaffolds*, *mode='blocks'*)

instagraal.parse_info_frags.**write_fasta**(*init_fasta*, *info_frags*, *output='new_genome.fa'*, *junction=False*)
Convert an info_frags.txt file into a fasta file given a reference. Optionally adds junction sequences to reflect the possibly missing base pairs between two newly joined scaffolds.

instagraal.parse_info_frags.**write_info_frags**(*scaffolds*, *output='new_info_frags.txt'*)

instagraal.pyramid_sparse module

Pyramid library

Create and handle so-called 'pyramid' objects, i.e. a series of decreasing-resolution contact maps in hdf5 format.

instagraal.pyramid_sparse.**abs_contact_2_coo_file**(*abs_contact_file*, *coo_file*)

Convert contact maps between old-style and new-style formats.

A legacy function that converts contact maps from the older GRAAL format to the simpler instaGRAAL format. This is useful with datasets generated by Hi-C box.

Parameters

- **abs_contact_file** (*str, file or pathlib.Path*) – The input old-style contact map.
- **coo_file** (*str, file, or pathlib.Path*) – The output path to the generated contact map; must be writable.

instagraal.pyramid_sparse.**build**(*base_folder*, *size_pyramid*, *factor*, *min_bin_per_contig*)

Build a pyramid of contact maps

Build a fragment pyramid for multi-scale analysis

Parameters

- **base_folder** (*str or pathlib.Path*) – Where to create the hdf5 files containing the matrices.
- **size_pyramid** (*int*) – How many levels (contact maps of decreasing resolution) to generate.
- **factor** (*int*) – Subsampling factor (binning) from one level to the next.
- **min_bin_per_contig** (*int*) – The minimum number of bins per contig below which binning shall not be performed.

instagraal.pyramid_sparse.**build_and_filter**(*base_folder*, *size_pyramid*, *factor*, *thresh_factor=1*)

Build a filtered pyramid of contact maps

Build a fragment pyramid for multi-scale analysis and remove high sparsity (i.e. low-coverage) and short fragments.

Parameters

- **base_folder** (*str or pathlib.Path*) – Where to create the hdf5 files containing the matrices.
- **size_pyramid** (*int*) – How many levels (contact maps of decreasing resolution) to generate.
- **factor** (*int*) – Subsampling factor (binning) from one level to the next.
- **thresh_factor** (*float, optional*) – Number of standard deviations below the mean coverage beyond which lesser covered fragments will be discarded. Default is 1.

Returns **obj_pyramid** – The pyramid object containing all the levels.

Return type Pyramid

instagraal.pyramid_sparse.**file_len**(*fname*)

instagraal.pyramid_sparse.**fill_sparse_pyramid_level**(*pyramid_handle*, *level*, *contact_file*, *nfrags*)

Fill a level with sparse contact map data

Fill values from the simple text matrix file to the hdf5-based pyramid level with contact data.

Parameters

- **pyramid_handle** (*h5py.File*) – The hdf5 file handle containing the whole dataset.
- **level** (*int*) – The level (resolution) to be filled with contact data.
- **contact_file** (*str, file or pathlib.Path*) – The binned contact map file to be converted to hdf5 data.
- **nfrags** (*int*) – The number of fragments/bins in that specific level.

instagraal.pyramid_sparse.**get_contig_info_from_file**(*contig_info*)

instagraal.pyramid_sparse.**get_frag_info_from_fil**(*fragments_list*)

instagraal.pyramid_sparse.**init_frag_list**(*fragment_list*, *new_frag_list*)

Adapt the original fragment list to fit the build function requirements

Parameters

- **fragment_list** (*str, file or pathlib.Path*) – The input fragment list.
- **new_frag_list** (*str, file or pathlib.Path*) – The output fragment list to be written.

Returns **i** – The number of records processed this way.

Return type int

class instagraal.pyramid_sparse.**level**(*pyramid*, *level*)

Bases: object

build_seq_per_bin(*genome_fasta*)

define_inter_chrom_coord()

generate_new_fasta(*vect_frags*, *new_fasta*, *info_frags*)

init_data()

load_data(*pyramid*)

Parameters **pyramid** – hic pyramid

instagraal.pyramid_sparse.**main**()

instagraal.pyramid_sparse.**new_remove_problematic_fragments**(*contig_info*, *fragments_list*, *abs_fragments_contacts*, *new_contig_list_file*, *new_fragments_list_file*, *new_abs_fragments_contacts_file*, *pyramid*)

class instagraal.pyramid_sparse.**pyramid**(*pyramid_folder*, *n_levels*)

Bases: object

build_frag_dictionnary(*fragments_list*, *level*)

close()

full_zoom_in_frag(*curr_frag*)

Parameters **curr_frag** –

get_level(*level_id*)

load_reference_sequence(*genome_fasta*)

update_super_index(*dict_frag*, *super_index_file*)

update_super_index_in_dict_contig(*dict_frag*, *dict_contig*)

zoom_in_area(*area*)
zoom in area

zoom_in_frag(*curr_frag*)

Parameters **curr_frag** –

zoom_in_pixel(*curr_pixel*)
return the curr_frag at a higher resolution

zoom_out_frag(*curr_frag*)

Parameters **curr_frag** –

zoom_out_pixel(*curr_pixel*)
return the curr_frag at a lower resolution

instagraal.pyramid_sparse.**remove_problematic_fragments**(*contig_info*, *fragments_list*, *abs_fragments_contacts*, *new_contig_list_file*, *new_fragments_list_file*, *new_abs_fragments_contacts_file*, *pyramid*, *thresh_factor=1*)

instagraal.pyramid_sparse.**subsample_data_set**(*contig_info*, *fragments_list*, *fact_sub_sample*, *abs_fragments_contacts*, *new_abs_fragments_contacts_file*, *min_bin_per_contig*, *new_contig_list_file*, *new_fragments_list_file*, *old_2_new_file*)

instagraal.simu_single module

instagraal.simu_single.**kth_diag_indices**(*a*, *k*)

class instagraal.simu_single.**simulation**(*name*, *folder_path*, *fasta*, *level*, *n_iterations*, *is_simu*, *gl_window*, *use_rippe*, *gl_size_im*, *thresh_factor=1*, *output_folder=None*)

Bases: object

blacklist_contig()

create_new_sub_frags()

create_sub_frags()

discard_low_coverage_frags()

export_new_fasta()

init_gl_image()

load_gl_buffers()

modify_sub_vect_frags()
include repeated frags

modify_vect_frags()
include repeated frags

plot_info_simu(*collect_likelihood_input*, *collect_n_contigs_input*, *file_plot*, *title_ax*)

release()

select_data_set(*name*)

select_repeated_frags()

instagraal.vector module

class instagraal.vector.**Vec**
Bases: numpy.ndarray

props = ['x', 'y', 'z', 'w']

instagraal.vector.**normalize**(*u*)

instagraal.version module

Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

i

instagraal, 23
instagraal.cuda_lib_gl_single, 3
instagraal.fragment, 5
instagraal.glutil, 6
instagraal.gpustruct, 6
instagraal.init_nuisance, 6
instagraal.instagraal, 6
instagraal.leastsqbound, 9
instagraal.linkage, 9
instagraal.log, 16
instagraal.optim_rippe_curve_update, 16
instagraal.parse_info_frags, 17
instagraal.pyramid_sparse, 20
instagraal.simu_single, 22
instagraal.vector, 23
instagraal.version, 23

Index

A

abs_contact_2_coo_file() (*in module instagraal.pyramid_sparse*), 20
apply_replay_simu() (*instagraal.cuda_lib_gl_single.sampler method*), 3
approx_all_likelihood_on_zeros() (*instagraal.cuda_lib_gl_single.sampler method*), 3
approx_single_likelihood_on_zeros() (*instagraal.cuda_lib_gl_single.sampler method*), 3
approx_single_likelihood_on_zeros_mut() (*instagraal.cuda_lib_gl_single.sampler method*), 3
approx_single_likelihood_on_zeros_nuisance() (*instagraal.cuda_lib_gl_single.sampler method*), 3

B

basic_fragment (*class in instagraal.fragment*), 5
blacklist_contig() (*instagraal.simu_single.simulation method*), 22
bomb_the_genome() (*instagraal.cuda_lib_gl_single.sampler method*), 4
build() (*in module instagraal.pyramid_sparse*), 20
build_and_filter() (*in module instagraal.pyramid_sparse*), 20
build_frag_dictionnary() (*instagraal.pyramid_sparse.pyramid method*), 21
build_seq_per_bin() (*instagraal.pyramid_sparse.level method*), 21

C

calc_cov_x() (*in module instagraal.leastsqbound*), 9
close() (*instagraal.pyramid_sparse.pyramid method*), 21
collapse_degenerate_markers() (*in module instagraal.linkage*), 10
compare_orderings() (*in module instagraal.linkage*), 10
copy() (*instagraal.fragment.fragment class method*), 5
copy_from_gpu() (*instagraal.gpustruct.GPUStruct method*), 6
copy_to_gpu() (*instagraal.gpustruct.GPUStruct method*), 6
correct_scaffolds() (*in module instagraal.parse_info_frags*), 17
correct_spurious_inversions() (*in module instagraal.parse_info_frags*), 17
create_gpu_struct() (*instagraal.cuda_lib_gl_single.sampler method*), 4
create_new_sub_frags() (*instagraal.simu_single.simulation method*), 22
create_sub_frags() (*instagraal.simu_single.simulation method*), 22
cuda_gl_init() (*instagraal.instagraal.window method*), 8

D

debug_test_EM() (*instagraal.instagraal.window method*), 8
debug_test_model() (*instagraal.instagraal.window method*), 8
define_inter_chrom_coord() (*instagraal.pyramid_sparse.level method*), 21
discard_low_coverage_frags() (*instagraal.simu_single.simulation method*), 22
display_current_matrix() (*instagraal.cuda_lib_gl_single.sampler method*), 4
dist_inter_genome() (*instagraal.cuda_lib_gl_single.sampler method*), 4
draw() (*instagraal.instagraal.window method*), 8
draw_axes() (*in module instagraal.glutil*), 6

draw_line() (*in module instagraal.glutil*), 6

E

err() (*in module instagraal.leastsqbound*), 9
estimate_max_dist_intra() (*in module instagraal.init_nuisance*), 6
estimate_max_dist_intra() (*in module instagraal.optim_rippe_curve_update*), 16
estimate_max_dist_intra_nuis() (*in module instagraal.optim_rippe_curve_update*), 16
estimate_param_hic() (*in module instagraal.init_nuisance*), 6
estimate_param_rippe() (*in module instagraal.optim_rippe_curve_update*), 16
estimate_parameters() (*instagraal.cuda_lib_gl_single.sampler method*), 4
estimate_parameters_rippe() (*instagraal.cuda_lib_gl_single.sampler method*), 4
eval_all_sub_likelihood() (*instagraal.cuda_lib_gl_single.sampler method*), 4
eval_likelihood() (*instagraal.cuda_lib_gl_single.sampler method*), 4
eval_likelihood_4_nuisance() (*instagraal.cuda_lib_gl_single.sampler method*), 4
eval_likelihood_init() (*instagraal.cuda_lib_gl_single.sampler method*), 4
eval_likelihood_on_mut() (*instagraal.cuda_lib_gl_single.sampler method*), 4
explode_genome() (*instagraal.cuda_lib_gl_single.sampler method*), 4
export_new_fasta() (*instagraal.simu_single.simulation method*), 22
external2internal() (*in module instagraal.leastsqbound*), 9
extract_current_sub_likelihood() (*instagraal.cuda_lib_gl_single.sampler method*), 4
extract_uniq_mutations() (*instagraal.cuda_lib_gl_single.sampler method*), 4

F

f_hic() (*instagraal.cuda_lib_gl_single.sampler method*), 4
f_rippe() (*instagraal.cuda_lib_gl_single.sampler method*), 4
file_len() (*in module instagraal.pyramid_sparse*), 20
fill_dist_all_mut() (*instagraal.cuda_lib_gl_single.sampler method*), 4
fill_dist_single() (*instagraal.cuda_lib_gl_single.sampler method*), 4
fill_dist_single_mut() (*instagraal.cuda_lib_gl_single.sampler method*), 4
fill_sparse_pyramid_level() (*in module instagraal.pyramid_sparse*), 20
find_lost_dna() (*in module instagraal.parse_info_frags*), 18
format_info_frags() (*in module instagraal.parse_info_frags*), 18
fragment (*class in instagraal.fragment*), 5
free_gpu() (*instagraal.cuda_lib_gl_single.sampler method*), 4
full_em() (*instagraal.instagraal.window method*), 8
full_zoom_in_frag() (*instagraal.pyramid_sparse.pyramid method*), 21

G

generate_new_fasta() (*instagraal.pyramid_sparse.level method*), 21
genome_content() (*instagraal.cuda_lib_gl_single.sampler method*), 4
get_contig_info_from_file() (*in module instagraal.pyramid_sparse*), 21
get_frag_info_from_fil() (*in module instagraal.pyramid_sparse*), 21
get_level() (*instagraal.pyramid_sparse.pyramid method*), 22
get_missing_blocks() (*in module instagraal.linkage*), 11
get_packed() (*instagraal.gpustruct.GPUStruct method*), 6
get_ptr() (*instagraal.gpustruct.GPUStruct method*), 6
glinit() (*instagraal.instagraal.window method*), 8
glut_print() (*instagraal.instagraal.window method*), 8
GPUStruct (*class in instagraal.gpustruct*), 6

I

i2e_cov_x() (*in module instagraal.leastsqbound*), 9
init() (*in module instagraal.glutil*), 6
init_data() (*instagraal.pyramid_sparse.level method*), 21

init_frag_list() (*in module instagraal.pyramid_sparse*), 21
init_gl_image() (*instagraal.simu_single.simulation method*), 22
initiate() (*instagraal.fragment.basic_fragment class method*), 5
initiate() (*instagraal.fragment.fragment class method*), 5
insert_blocks() (*instagraal.cuda_lib_gl_single.sampler method*), 4
insert_repeats() (*instagraal.cuda_lib_gl_single.sampler method*), 4
instagraal (*module*), 23
instagraal.cuda_lib_gl_single (*module*), 3
instagraal.fragment (*module*), 5
instagraal.glutil (*module*), 6
instagraal.gpustruct (*module*), 6
instagraal.init_nuisance (*module*), 6
instagraal.instagraal (*module*), 6
instagraal.leastsqbound (*module*), 9
instagraal.linkage (*module*), 9
instagraal.log (*module*), 16
instagraal.optim_rippe_curve_update (*module*), 16
instagraal.parse_info_frags (*module*), 17
instagraal.pyramid_sparse (*module*), 20
instagraal.simu_single (*module*), 22
instagraal.vector (*module*), 23
instagraal.version (*module*), 23
integrate_lost_dna() (*in module instagraal.parse_info_frags*), 18
internal2external() (*in module instagraal.leastsqbound*), 9
internal2external_grad() (*in module instagraal.leastsqbound*), 9
is_block() (*in module instagraal.parse_info_frags*), 18

K

kth_diag_indices() (*in module instagraal.simu_single*), 22

L

leastsqbound() (*in module instagraal.leastsqbound*), 9
level (*class in instagraal.pyramid_sparse*), 21
lights() (*in module instagraal.glutil*), 6
linkage_group_ordering() (*in module instagraal.linkage*), 13
load_data() (*instagraal.pyramid_sparse.level method*), 21
load_gl_buffers() (*instagraal.simu_single.simulation method*), 22
load_gl_cuda_tex_buffer() (*instagraal.cuda_lib_gl_single.sampler method*), 4
load_gl_cuda_vbo() (*instagraal.cuda_lib_gl_single.sampler method*), 4
load_reference_sequence() (*instagraal.pyramid_sparse.pyramid method*), 22
loadProgram() (*instagraal.cuda_lib_gl_single.sampler method*), 4
local_flip() (*instagraal.cuda_lib_gl_single.sampler method*), 4
log_peval() (*in module instagraal.optim_rippe_curve_update*), 16
log_residuals() (*in module instagraal.init_nuisance*), 6
log_residuals() (*in module instagraal.optim_rippe_curve_update*), 16
log_residuals_4_min() (*in module instagraal.init_nuisance*), 6

M

main() (*in module instagraal.instagraal*), 7
main() (*in module instagraal.linkage*), 13
main() (*in module instagraal.parse_info_frags*), 18
main() (*in module instagraal.pyramid_sparse*), 21
modify_genome() (*instagraal.cuda_lib_gl_single.sampler method*), 4
modify_gl_cuda_buffer() (*instagraal.cuda_lib_gl_single.sampler method*), 4
modify_image_thresh() (*instagraal.cuda_lib_gl_single.sampler method*), 4
modify_image_thresh() (*instagraal.instagraal.window method*), 8
modify_param_simu() (*instagraal.cuda_lib_gl_single.sampler method*), 4
modify_sub_vect_frags() (*instagraal.simu_single.simulation method*), 22
modify_vect_frags() (*instagraal.simu_single.simulation method*), 23

N

new_remove_problematic_fragments() (*in module instagraal.pyramid_sparse*), 21
normalize() (*in module instagraal.vector*), 23

O

on_click() (*instagraal.instagraal.window method*), 8
on_key() (*instagraal.instagraal.window method*), 8
on_mouse_motion() (*instagraal.instagraal.window method*), 8

P

parse_bed() (*in module instagraal.parse_info_frags*), 18
parse_info_frags() (*in module instagraal.parse_info_frags*), 18
parse_linkage_csv() (*in module instagraal.linkage*), 13
perform_mutations() (*instagraal.cuda_lib_gl_single.sampler method*), 4
peval() (*in module instagraal.init_nuisance*), 6
peval() (*in module instagraal.optim_rippe_curve_update*), 16
plot_info_frags() (*in module instagraal.parse_info_frags*), 18
plot_info_simu() (*instagraal.simu_single.simulation method*), 23
pop_out_pop_in() (*instagraal.cuda_lib_gl_single.sampler method*), 4
prepare_sparse_call() (*instagraal.cuda_lib_gl_single.sampler method*), 5
props (*instagraal.vector.Vec attribute*), 23
pyramid (*class in instagraal.pyramid_sparse*), 21

R

rearrange_intra_scaffolds() (*in module instagraal.parse_info_frags*), 19
release() (*instagraal.simu_single.simulation method*), 23
remote_update() (*instagraal.instagraal.window method*), 8
remove_problematic_fragments() (*in module instagraal.pyramid_sparse*), 22
remove_spurious_insertions() (*in module instagraal.parse_info_frags*), 19
render() (*instagraal.instagraal.window method*), 8
render_image() (*instagraal.instagraal.window method*), 8
reorient_consecutive_blocks() (*in module instagraal.parse_info_frags*), 19
replay_simu() (*instagraal.instagraal.window method*), 8
residual_4_max_dist() (*in module instagraal.init_nuisance*), 6
residual_4_max_dist() (*in module instagraal.optim_rippe_curve_update*), 16
residuals() (*in module instagraal.init_nuisance*), 6
residuals() (*in module instagraal.optim_rippe_curve_update*), 17
return_neighbours() (*instagraal.cuda_lib_gl_single.sampler method*), 5
return_rippe_vals() (*instagraal.cuda_lib_gl_single.sampler method*), 5

S

sampler (*class in instagraal.cuda_lib_gl_single*), 3
save_behaviour_to_txt() (*instagraal.instagraal.window method*), 8
select_data_set() (*instagraal.simu_single.simulation method*), 23
select_repeated_frags() (*instagraal.simu_single.simulation method*), 23
set_jumping_distributions_parameters() (*instagraal.cuda_lib_gl_single.sampler method*), 5
setup_all_gpu_struct() (*instagraal.cuda_lib_gl_single.sampler method*), 5
setup_distri_frags() (*instagraal.cuda_lib_gl_single.sampler method*), 5
setup_model_parameters() (*instagraal.cuda_lib_gl_single.sampler method*), 5
setup_rippe_parameters() (*instagraal.cuda_lib_gl_single.sampler method*), 5
setup_rippe_parameters_4_simu() (*instagraal.cuda_lib_gl_single.sampler method*), 5
setup_simu() (*instagraal.instagraal.window method*), 8
setup_thrust_modules() (*instagraal.cuda_lib_gl_single.sampler method*), 5
show_sub_slice() (*instagraal.cuda_lib_gl_single.sampler method*), 5
simple_start() (*instagraal.instagraal.window method*), 8
simulation (*class in instagraal.simu_single*), 22
slice_sparse_mat() (*instagraal.cuda_lib_gl_single.sampler method*), 5
sparse_data_2_gpu() (*instagraal.cuda_lib_gl_single.sampler method*), 5
sparse_data_4_gl() (*insta-*

graal.cuda_lib_gl_single.sampler method), 5
start_EM() (*instagraal.instagraal.window method*), 8
start_EM_all() (*instagraal.instagraal.window method*), 8
start_EM_no_scrambled() (*instagraal.instagraal.window method*), 8
start_EM_nuisance() (*instagraal.instagraal.window method*), 8
start_MCMC() (*instagraal.instagraal.window method*), 8
start_MTM() (*instagraal.instagraal.window method*), 8
step_nuisance_parameters() (*instagraal.cuda_lib_gl_single.sampler method*), 5
step_sampler() (*instagraal.cuda_lib_gl_single.sampler method*), 5
step_sampler_debug() (*instagraal.cuda_lib_gl_single.sampler method*), 5
subsample_data_set() (*in module instagraal.pyramid_sparse*), 22

T

temperature() (*instagraal.cuda_lib_gl_single.sampler method*), 5
test_copy_struct() (*instagraal.cuda_lib_gl_single.sampler method*), 5
test_model() (*instagraal.instagraal.window method*), 8
test_thrust() (*instagraal.cuda_lib_gl_single.sampler method*), 5
timer() (*instagraal.instagraal.window method*), 8
transloc() (*instagraal.cuda_lib_gl_single.sampler method*), 5

U

update_name() (*instagraal.fragment.fragment method*), 5
update_neighbourhood() (*instagraal.cuda_lib_gl_single.sampler method*), 5
update_super_index() (*instagraal.pyramid_sparse.pyramid method*), 22
update_super_index_in_dict_contig() (*instagraal.pyramid_sparse.pyramid method*), 22

V

Vec (*class in instagraal.vector*), 23

W

window (*class in instagraal.instagraal*), 7
write_bed() (*in module instagraal.linkage*), 16
write_fasta() (*in module instagraal.parse_info_frags*), 19
write_info_frags() (*in module instagraal.parse_info_frags*), 19

Z

zoom_in_area() (*instagraal.pyramid_sparse.pyramid method*), 22
zoom_in_frag() (*instagraal.pyramid_sparse.pyramid method*), 22
zoom_in_pixel() (*instagraal.pyramid_sparse.pyramid method*), 22
zoom_out_frag() (*instagraal.pyramid_sparse.pyramid method*), 22
zoom_out_pixel() (*instagraal.pyramid_sparse.pyramid method*), 22