Pinetree Documentation

Release 0.0.1

Benjamin Jack

May 28, 2018

User Documentation

Pinetree is a stochastic gene expression simulator that tracks polymerases and ribosomes at the single-molecule level. It includes nucleotide-resolution transcription and translation rates. This granularity allows users to simulate the effects of codon-usage bias and dinucleotide bias on gene expression.

Pinetree is a stochastic simulation that is implemented in Python with a C++ back-end and designed to be highly efficient. On a desktop-class CPU, simulating gene expression in a 40 kilobase viral genome for 30 minutes takes about 3 hours of computation time.

CHAPTER 1

Introduction

1.1 Installation

Pinetree has no requirements except Python and CMake. Python 3 is recommend. To install pinetree from pip, run the following:

```
pip install pinetree
```

The latest development build may be installed from GitHub:

```
git clone https://github.com/benjaminjack/pinetree.git
pinetree/setup.py install
```

1.2 Construct a simulation

All pinetree simulations begin with the construction of a Model object. At a minimum, Model must define the volume in which the simulation will take place.

```
import pinetree as pt

model = pt.Model(cell_volume=8e-16)
```

Next, we'll define a genome and register it with Model.

1.3 Defining a genome

Pinetree supports linear genomes of any size, represented by Genome objects. A Genome object must be given a name and a length, in base pairs.

```
plasmid = pt.Genome(name="myplasmid", length=300)
```

After defining a Genome object, we can add promoters, terminators, and genes. These genetic elements can be defined in any order.

```
plasmid.add_promoter(name="phi1", start=1, stop=10,
                     interactions={"rnapol": 2e8})

plasmid.add_terminator(name="t1", start=299, stop=300,
                       efficiency={"rnapol": 1.0})

plasmid.add_gene(name="rnapol", start=26, stop=225,
                 rbs_start=11, rbs_stop=26, rbs_strength=1e7)

plasmid.add_gene(name="proteinX", start=241, stop=280,
                 rbs_start=226, rbs_stop=241, rbs_strength=1e7)
```

Here we've defined a plasmid with two genes, 'rnapol', an RNA polymerase which binds to promoter 'phi1' and some other 'proteinX'. Each genetic element has a name, a start position, and a stop position. For more information on the other arguments of these methods, please see the full Python documentation.

When all genetic elements have been added, register the Genome object with our Model object.

```
model.register_genome(plasmid)
```

At this point we could run the simulation, but nothing would happen because we have not defined any polymerases or ribosomes that interact with the Genome object.

1.4 Defining polymerases and ribosomes

To simulate both transcription and translation, we'll add polymerases and then add ribosomes. Since these enzymes may interact with more than one type of genome, we add them to the Model object.

```
model.add_polymerase(name="rnapol", speed=40, footprint=10, copies=10)
model.add_ribosome(speed=30, footprint=10, copies=100)
```

Polymerases and ribosomes may move at any speed. Their respective footprints, however, must be smaller than the sites to which they bind. For example, if 'rnapol' has a footprint of 10 bp, then the promoter it binds to must also be at least 10 bp in length. Likewise, if a ribosome has a footprint of 10 bp, the ribosome binding site must be at least 10 bp.

1.5 Define species reactions

Pinetree supports option species reactions between one or two molecular species. For example, we may define a reaction such that proteinX forms a complex with rnapol called rnapol-X.

```
model.add_reaction(reactants=['proteinX', 'rnapol'],
                   products=['rnapol-X'],
                   rate=1e-7)
```

1.6 Run the simulation

To simulate gene expression, specify a time limit and a time step at which to output data. All protein and transcript counts will be output in tab seperated format.

```
model.simulate(time_limit=60, time_step=1, output="simulation.tsv")
```

1.7 Interpretting results

A pinetree simulation produces an output file with 5 columns.

time Current time of simulation in seconds.

species Name of a molecular species, derived from a polymerase name, a gene name, or an explicitly defined molecular species. Any name with a '__' double underscore prefix is used internally by pinetree.

protein Quantity of *free* proteins corresponding to a species name. For example, the number in this column corresponding to 'rnapol' would represent free RNA polymerases that are not actively transcribing.

transcript Quantity of transcripts for corresponding to a species name. If a species only exists as a protein or otherwise has no transcript precursor, this value will be 0.

ribo_density (experimental) Average quantity of ribosomes actively translating on a transcript.

CHAPTER 2

Python reference

This is an reference for the public python pinetree interface.

2.1 Model

class `pinetree.`**`Model`**(*self: pinetree.core.Model*, *cell_volume: float*) → None

Define a pinetree model.

Parameters **`cell_volume`** (*`float`*) – The volume, in liters, of the system being simulated.

Examples

```
>>> import pinetree.pinetree as pt
>>> sim = pt.Model(cell_volume=8e-16) # Approximate volume of E. coli cell
```

`add_polymerase`(*self: pinetree.core.Model*, *name: str*, *footprint: int*, *speed: float*, *copy_number: int*) → None

Add a polymerase to the model. There may be multiple types of polymerases in a model.

Note: Defining a polymerase with a footprint larger than that of the promoter it binds is not currently supported.

Parameters

- **`name`** (*`str`*) – Name of the polymerase which can be referred to in `add_reaction()` and `add_promoter()`.
- **`copy_number`** (*`int`*) – Initial number of copies of the polymerase
- **`speed`** (*`int`*) – Speed, in base pairs per second, at which the polymerase transcribes
- **`footprint`** (*`int`*) – Footprint, in base pairs, of the polymerase on the genome

add_reaction(*self: pinetree.core.Model, rate_constant: float, reactants: List[str], products: List[str]*) → None

Define a reaction between species, which may include free ribosomes and polymerases.

Parameters

- **rate_constant** (*float*) – Macroscopic rate constant of the reaction. This will be converted into a stochastic mesoscopic rate constant automatically.
- **reactants** (*list*) – List of reactants, which may be species, ribosomes, or polymerases
- **products** (*list*) – List of products which may be species, ribosomes, or polymerases

Note: Reaction rate constants should be given as macroscopic rate constants, the same constants used in differential equation-based models. Pinetree will automatically convert these rate constants to mesoscopic constants required for a stochastic simulation.

Example

```
>>> coming soon
```

add_ribosome(*self: pinetree.core.Model, footprint: int, speed: float, copy_number: int*) → None

Add ribosomes to the model. There may only be a single type of ribosome.

Note: Defining ribosomes with a footprint larger than that of the promoter it binds is not currently supported.

Parameters

- **copy_number** (*int*) – Initial number of copies of free ribosomes
- **speed** (*int*) – Mean speed, in base pairs per second, at which the ribosome translates. This speed will be scaled on a per site basis if translation weights are defined. (See Genome.AddWeights).
- **footprint** (*int*) – Footprint, in base pairs, of the ribosome on RNA

add_species(*self: pinetree.core.Model, name: str, copy_number: int*) → None

Defines individual chemical species not specified by either `add_ribosome()` or `add_polymerase()`.

Parameters

- **name** (*str*) – Name of chemical species which can be referred to in reactions added with `add_reaction()`.
- **copy_number** (*int*) – Initial number of copies of the chemical species

register_genome(*self: pinetree.core.Model, arg0: Genome*) → None

Register a genome with the model.

Parameters **genome** (`Genome`) – a pinetree `Genome` object.

seed(*self: pinetree.core.Model, arg0: int*) → None

Set a seed for reproducible simulations.

Parameters **seed** (*int*) – a seed for the random number generator

simulate (*self: pinetree.core.Model*, *time_limit: int*, *time_step: int*, *output: str='counts.tsv'*) → None
Run a gene expression simulation. Produces a tab separated file of protein and transcript counts at user-specified time intervals.

Parameters

- **time_limit** (*int*) – Simulated time, in seconds at which this simulation should stop executing reactions. Note that this *simulated* time and not real time. The real time that it takes for the simulation to complete depends on the number of reactions and species (genomes, transcripts, proteins, etc) in the system.
- **time_step** (*int*) – Time interval, in seconds, that species counts are reported.
- **output** (*str*) – Name of output file (default: counts.tsv).

2.2 Genome

class pinetree.**Genome** (*self: pinetree.core.Genome*, *name: str*, *length: int*, *transcript_degradation_rate: float=0.0*, *rnase_speed: float=0.0*, *rnase_footprint: int=0*) → None
Define a linear genome.

Warning: Transcript degradation is an experimental feature. Defining `transcript_degradation_rate`, `rnase_speed`, or `rnase_footprint` may crash pinetree.

Parameters

- **name** (*str*) – Name of genome.
- **length** (*int*) – Length of genome in base pairs.
- **transcript_degradation_rate** (*float*) – Unary binding rate constant for binding of RNases to RNase sites.
- **rnase_speed** (*flaot*) – Mean speed at which RNase degrades transcript, in bases per second.
- **rnase_footprint** (*float*) – Initial footprint of RNase on RNA.

add_gene (*self: pinetree.core.Genome*, *name: str*, *start: int*, *stop: int*, *rbs_start: int*, *rbs_stop: int*, *rbs_strength: float*) → None
Define a gene. Genes may be defined in any order.

Note: At this time, overlapping genes or genes that overlap with ribosome binding sites are not supported.

Parameters

- **name** (*str*) – Name of gene. Name may be referenced by `Genome.add_reaction`.
- **start** (*int*) – Start position of gene.
- **stop** (*int*) – Stop position of gene.

- **rbs_start** (*int*) – Start position of ribosome binding site. Generally positioned upstream of gene start.
- **rbs_stop** (*int*) – Stop position of ribosome binding site.
- **rbs_strength** (*float*) – Binding rate constant between ribosome and ribosome binding site.

add_mask (*self: pinetree.core.Genome, start: int, interactions: List[str]*) → None

Mask a portion of this Genome. This mask may correspond to a portion of the genome that has not yet entered the cell or is otherwise inaccessible. Also define which Polymerases are capabile of moving the Mask (e.g. an RNA polymerase that actively pulls the genome into the cell.)

Parameters

- **start** (*int*) – Start position of Mask. The Mask is assumed to extend to the end of the genome.
- **interactions** (*list*) – List of Polymerase names capable of shifting the Mask backwards and revealing more of the genome.

add_promoter (*self: pinetree.core.Genome, name: str, start: int, stop: int, interactions: Dict[str, float]*) → None

Define a promoter.

Parameters

- **name** (*str*) – Name of promoter.
- **start** (*int*) – Start position of promoter.
- **stop** (*int*) – Stop position of promoter.
- **interactions** (*dict*) – Dictionary of binding rate constants for different Polymerases that this promoter interacts with.

Example

```
>>> genome.add_promoter(name="phi1", start=1, stop=10,
>>>                     interactions={'rnapol': 1e7})
```

add_rnase_site (*self: pinetree.core.Genome, start: int, stop: int*) → None

Define an internal RNase cleavage site.

Warning: This feature is experimental and adding RNase cleavage sites may crash pinetree.

Parameters

- **start** (*int*) – Start position of RNase cleavage site.
- **stop** (*int*) – Stop position of RNase cleavage site.

add_terminator (*self: pinetree.core.Genome, name: str, start: int, stop: int, efficiency: Dict[str, float]*) → None

Define a terminator.

Parameters

- **name** (*str*) – Name of terminator.

- **`start`** (*`int`*) – Start position of terminator.
- **`stop`** (*`int`*) – Stop position of terminator.
- **`efficiency`** (*`dict`*) – Dictionary of termination efficiencies (between 0 and 1) for different Polymerases that this terminator interacts with. A value of 1.0 represents complete stop of transcription and removal of the Polymerase. A value of 0.0 means that the Polymerase will always read through the terminator and continue transcription.

Example

```
>>> genome.add_terminator(name="t1", start=50, stop=51,
>>>                          efficiency={'rnapol': 0.85})
```

`add_weights`(*self: pinetree.core.Genome, weights: List[float]*) → None

Define position-specific translation speed weights. These may correspond, for example, codon-specific translation rates.

Parameters **`weights`** (*`list`*) – List of weights of same length as Genome. These weights are multiplied by the ribosome speed to calculate a final translation rate at every position in the genome.

CHAPTER 3

Design and Implementation

The purpose of this document is to describe, at a conceptual level, the internal structure and logic of Pinetree. If any sections seem confusing or unclear, please file an issue on GitHub.

3.1 Stochastic Simulation Algorithm

Pinetree employs the Gillespie Stochastic Model Algorithm (SSA) to model all molecular interactions involved in gene expression, including the movement of individual polymerases on DNA and ribosomes on mRNA. In the SSA, we consider all molecular interactions as reactions, extending from a parent Reaction class. The Gillespie class contains the core components of the SSA. The Gillespie SSA is defined as follows:

1. **Initialize**. Begin with time $t = 0$. Specify free species and assign them a copy number. Specify a set of reactions and reaction rate constants involving the free species.
2. **Compute propensities**. For all reactions in the system compute the propensity of the reaction occurring. Sum over all propensities for all reactions to get the total propensity.
3. **Generate random numbers**. Based on the individual propensities computed in step 2, randomly select a reaction to occur from all reactions in the system. Using the sum of all propensities, τ, compute the time that the next reaction is expected to occur.
4. **Execute**. Execute the reaction from step 4, and advance the time :math:t by τ.
5. **Iterate**. Repeat steps 2–5 until t exceeds some predefined end time of the simulation.

Pinetree follows these steps, while defining several specialized reactions that provide single-molecule tracking of polymerases and ribosomes along DNA and mRNA. We will now describe these specialized reactions, and each step of the SSA, in more detail.

3.2 Pooled species-level reactions

An abstract Reaction class is the parent class of all reactions in the SSA. Gillespie maintains a vector of all reactions in the SSA, and they must inherit from Reaction (Fig. 3.1).

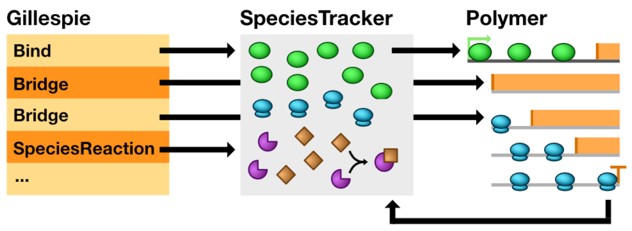

Fig. 3.1: Relationship between Model, SpeciesTracker, and individual Polymer objects. Upon execution, the Bind reaction constructs a Polymerase on a Polymer and removes one copy from SpeciesTracker. A Bridge reaction bypasses SpeciesTracker entirely and signals an individual Polymer to move a Polymerase. When a Polymerase reaches a Terminator, the parent Polymer destroys the Polymerase object and increases the copy number count of that Polymerase type in SpeciesTracker. Lastly, SpeciesReaction objects interact only with pooled species in SpeciesTracker.

To comply with the SSA framework, all reactions must be capable of three actions: calculating their propensity (Step 2) and executing (Step 4). SpeciesReaction implements a standard SSA reaction. SpeciesReaction defines a set of products, a set of reactants, and a rate constant, and computes its propensity from the copy numbers of interacting species and a rate constant. The SSA requires a mesoscopic rate constant, which differs from a macroscopic rate constant typical of deterministic models in that it depends on the reaction volume. SpeciesReaction converts macroscopic rate constants to mesoscopic rate constants internally. Thus the same rate constants from deterministic models can be used directly to parameterize Pinetree simulations.

Upon execution, SpeciesReaction increments its product counts and decrements the reactant counts. Pinetree only supports coefficients of one for reactants. Product coefficients can be any value greater than zero. SpeciesReaction objects themselves do not maintain counts of molecular species. All molecular species in Pinetree simulation run are tracked by a single instance of SpeciesTracker. SpeciesTracker also maintains species-to-SpeciesReaction maps. This allows Gillespie to cache propensities, and only update the propensities of reactions whose products or reactants have just changed.

3.3 Individual molecule-level reactions

At the single-molecule level, Polymer objects track individual Polymerases as they move along the polymer, detecting collisions and determining when the Polymerase should leave the Polymer. Tracking RNA polymerases on a genome and ribosomes on mRNA transcripts share enough similarities that most of transcription and translation logic is defined in a generic Polymer class. In practice, a Pinetree Model object uses the child classes Genome and Transcript, described in later sections. Ribosomes and RNA polymerases differ only in the definition of their member variables (e.g. footprint size, movement rate, step size, and binding interactions) and are thus represented only by the Polymerase class. Likewise, promoters and ribosome binding sites are represented by a single Promoter class and terminators and stop codons by a single Terminator class. The remainder of this section describes the Polymer, Polymerase, Promoter, and Terminator classes generically.

Pinetree defines two specialized reactions that differ from SpeciesReaction to handle single-molecule tracking. The first are called Bind reactions. Bind reactions coordinate the transfer of a Polymerases from the pooled species level to an individual Polymer object. Bind reactions are specific for each type Promoter-Polymerase binding interaction. Bind treats all exposed Promoters as a pooled species. Upon execution of the Bind reaction, Bind randomly selects a Polymer that contains an open Promoter with which to bind. It constructs a new Polymerase, then instructs the Polymer to bind the Polymerase to an open Promoter. SpeciesTracker maintains Promoter-to-Polymer maps. This map allows all components of Pinetree to quickly look up which Polymers contain a given Promoter.

The second type of specialized single-molecule reactions are called Wrapper reactions. Wrapper reactions are thin wrappers around individual Polymers. Every individually modeled Polymer has an associated Wrapper reaction. The Wrapper reaction requests a total propensity value from its Polymer. Upon execution, it instructs the Polymer to move one of its Polymerases.

3.3.1 Polymer

Each Polymer object maintains a vector of Polymerase objects attached to the Polymer, ordered by position. Each Polymerase defines a fixed speed, in base pairs per second, of movement. Each Polymer also maintains a vector of scaling factors representing every base in the Polymer. To simulate codons that are translated at different speeds, Polymer computes position-specific movement rates. To compute the scaled movement rates for all Polymerase objects, the movement rate of Polymerase is multiplied by a scaling factor corresponding to the position of the Polymer. These scaled movement rates correspond exactly to SSA propensities. Thus, the sum of these propensities represent the overall propensity of the Polymer, i.e., a value proportional to the probability of any single Polymerase moving along the Polymer. This overall propensity is reported to Gillespie, which tracks all Polymers via Wrapper reactions. If Gillespie selects a Wrapper reaction to execute corresponding to a given Polymer, that Polymer must then choose which Polymerase to move. To move a Polymerase, the Polymer again takes the scaled movement rates (propensities) and selects a Polymerase randomly, weighted by its propensity. The Polymer attempts to move that Polymerase one position forward.

Polymer checks for several interactions before Polymerase movement is finalized. First, it checks for a collision with a downstream Polymerase objects by comparing coordinates of the newly-moved Polymerase an upstream Polymerase. If the coordinates overlap, a collision has a occurred. The newly-moved Polymerase moves back to its original starting position, and the current SSA iteration ends. If no collision occurs, Polymer compares the coordinates of the newly-moved Polymerase with that of any upstream Terminator, Promoter, or Mask objects. If the Polymerase overlaps with Terminator, the Polymer verifies that the Polymerase interacts with the Terminator. To simulate readthrough, the Polymer randomly generates a number between 0 and 1. If this value is larger than the the readthrough probability of the Terminator, the Polymer finalizes the movement of Polymerase and sets a readthrough flag. This readthrough flag stops Polymer from repeatedly verifying Terminator-Polymerase interactions as the Polymerase moves over the terminator during future SSA iterations. Once Polymerase clears the Terminator completely, Polymer resets the readthrough flag. If no readthrough occurs, Polymer terminates Polymerase by removing the object from the vector of Polymerase objects. The Polymer recomputes its total propensity and fires a termination signal to other components of the simulation. This termination signal varies for Genome and Transcript described below.

If Polymerase overlaps with a Promoter, Polymer marks that Promoter as covered and inaccessible. Once Polymerase clears the Promoter, Polymer marks the Promoter as accessible again. Polymer maintains a vector of unbound Promoter objects, and SpeciesTracker maintains a map of which Promoter objects bind to which Polymerase objects. The Polymer reports to SpeciesTracker and Gillespie the number of unbound Promoter objects. If the Model determines that a Polymerase should bind to a Promoter, the Polymer randomly selects the appropriate Promoter to bind, and the polymerase is added to the vector of Polymerases at the Promoter object's position. The newly-bound Polymerase is now ready to move on the Polymer.

Lastly, Polymerase objects may interact with Mask objects upon moving. Each Polymer may have a single Mask object. The Mask objects makes portions of the Polymer inaccessible to Polymerases. Polymer treats the Mask as a large Polymerase that may cover the entire Polymer. Upon Polymerase movement, if the Polymerase collides with a Mask, the Polymerase may move back one step, or the Mask may recede. Which of these two interactions occurs depends on the specific Mask and Polymerase, and these interactions differ for Genome and Transcript objects.

3.3.2 Genome and Transcript

The Genome and Transcript classes are specialized versions of the parent Polymer class (Fig. 3.2).

A Genome object has a vector member variable that defines a complete transcript template. When a polymerase binds to a promoter, it immediately generates a complete Transcript object based on the transcript template. The newly-generated Transcript object contains genes corresponding to where the polymerase bound and extending to the end of the genome. Upon binding, the polymerase creates a Mask covering the entire Transcript, except for the very 5' end. As the polymerase moves forward from this promoter in the 5'-to-3' direction on the Genome

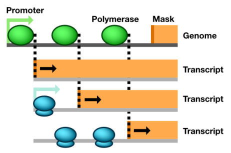

Fig. 3.2: Single molecule tracking in Pinetree. When a Polymerase binds to a Promoter, it immediately generates a Transcript object with a Mask. As the Polymerase moves, the Mask object retracts, exposing Promoters on the Transcripts. Dashed lines represent signals between Polymerase and Mask objects.

object, the polymerase signals to the Transcript to shift the 5'-end of Mask one base pair towards the 3'-end, thus exposing more of the Transcript. This unmasking process simulates transcript synthesis. Moreover, the length of the transcript, corresponding to the position of the mask, can be determined dynamically as the simulation progresses. The termination position of the transcript does not need to be specified upon promoter binding.

The Mask in a Transcript is inaccessible to ribosomes. Ribosomes, represented by Polymerase objects, collide with the Mask in much the same way that they collide with each other. A ribosome colliding with a Mask simulates ribosomes colliding with an RNA polymerase that is actively synthesizing the transcript on which the ribosome is translating. If a ribosome collides with a Mask, the ribosome stalls, just as if it had collided with another ribosome.

A Genome may also define a Mask. This Mask makes portions of the Genome inaccessible to polymerase binding. However, some polymerases are capable of shifting the mask upon colliding with it. This shifting simulates some viral genomes in which a polymerase itself pulls the genome into a cell as it transcribes.

3.4 FixedElements

FixedElement objects are defined as any fixed element along a Polymer. These include promoters, terminators, ribosome binding sites, and stop codons. FixedElement objects may interact with any number of different types of Polymerases. They are also capable of being covered by a Polymerase and thus inaccessible. All FixedElement objects differ from Polymerase and Mask objects in that they have fixed stop and start coordinates.

3.5 Signaling mechanisms

A Signal class provides a standardized interface for communication between different objects in Pinetree. For example, when a Polymerase moves it may signal to a transcript Mask that it should also move. When a ribosome reaches a stop codon, it signals to SpeciesTracker that a termination event has occurred and new protein must be added to the species pool. The Signal class follows a signals and slots" model. Some objects carry their own Signal objects. Any function from any object may register with the Signal and occupy one of the Signal slots. These slots represent listeners. An object may then fire a Signal object, transmitting signals to any number of listeners without knowing how such signals will be handled when they reach the listener. This encapsulation allows portions of the simulation to be tested independently from one another.

CHAPTER 4

Frequently asked questions

4.1 Do genomic coordinates start from 1 or from 0? Are start and stop coordinates inclusive?

Genomic coordinates *start from 1* and start and stop coordinates are *inclusive*. These conventions follow those established by the GenBank format. Internally, Pinetree converts these coordinates to a 0-based indices, but the end user need not worry about these conversions.

4.2 How do I specify transcript degradation rates?

Pinetree does not currently support transcript degradation. Any transcript produced during a simulation never degrades.

4.3 Where did the name come from?

The name Pinetree is a reference to the "Christmas tree-like" electron micrographs of transcription and translation. Strands of mRNA transcripts branch out from the DNA, and each transcript is studded with ribosomes. This pattern looks like a Christmas tree.

CHAPTER 5

C++ reference

namespace Random

Functions

static std::uniform_real_distribution Random::dis_(0, 1)

void *Random*`seed`(int *seed*)

double *Random*`random`()

template <typename T>
int *Random*`WeightedChoiceIndex`(**const** std::vector<T> &*population*, **const** std::vector<double> &*weights*)

template <typename T>
T *Random*`WeightedChoice`(**const** std::vector<T> &*population*, **const** std::vector<double> &*weights*)

template <typename T>
T *Random*`WeightedChoice`(**const** std::vector<T> &*population*)

Variables

bool *Random*`seeded_` = false

std::mt19937 *Random*`gen_`

CHAPTER 6

Indices and tables

- genindex
- search

Index