

1 QUETZAL - an open source C++ template library for 2 coalescence-based environmental demogenetic models inference

3 Arnaud Becheler, Camille Coron, Stéphane Dupas

4 November 2, 2017

5 **1 Abstract**

6 The purpose of this article is to introduce an implementation framework enabling us, using available genetic
7 samples, to understand and foresee the behavior of species living in a fragmented and temporally changing
8 environment. To this aim, we first present a model of coalescence which is conditioned to environment,
9 through an explicit modeling of population growth and migration. The parameters of this model can be
10 inferred using Approximate Bayesian Computation techniques, which supposes that the considered model can
11 be efficiently simulated. We next present Quetzal, a C++ library composed of reusable generic components
12 and designed to efficiently implement a wide range of coalescence-based environmental demogenetic models.

13 **2 Introduction**

14 **2.1 Motivations**

15 Understanding how species react to spatio-temporal environmental heterogeneity and how this conditions
16 the patterns of genetic variation is of great importance in the context of conservation biology, for example
17 to predict future species distributions under global climate change (Pauls et al., 2013). Spatially explicit
18 simulation studies have proven to be of fundamental importance when tackling such dynamical processes,
19 especially in the context of range expansions (Excoffier et al., 2009). Despite a growing number of simulation
20 programs dedicated to coalescence-based models of genetic variation, code reuse is still limited. We present
21 Quetzal, a new C++ library with reusable generic components designed to ease the implementation of a
22 wide range of coalescence-based environmental demogenetic models, and to embed the simulation in an

23 Approximate Bayesian Computation (ABC) framework. The code is open-source, and available at <https://github.com/Becheler/quetzal> (see Becheler, 2017).

25 **2.2 Context**

26 Present genetic data can be linked to past ecological processes by coupling demographic models accounting
27 for the spatio-temporal landscape heterogeneity with models of genetic variation (see Figure 1). When the
28 studied genetic variation is neutral, genetic models based on coalescent approaches (Nordborg, 2001; Hein
29 et al., 2004; Wakeley, 2009) can be used. In this framework, the coalescence of two gene copies into a parent
30 copy is simply the replication of the ADN viewed backward in time. The genealogy of the sampled genes
31 copies can be defined backward in time conditionally to the demographic process which itself can be defined
32 before tackling genetical aspects. This is an important theoretical link between a genetic sample and the
33 historical processes that shaped it, and it can be used for constructing statistical models allowing to estimate
34 properties of these past processes on the basis of the present sample.

35 Constructing such estimates often relies on the study of the likelihood, that gives the probability of
36 data to arise, as a function of the parameter θ of the statistical model. The likelihood function can be
37 derived under simple models, but as theoretical advances steered models towards higher levels of complexity
38 (migration (Beerli and Felsenstein, 1999), recombination (Kuhner et al., 2000), selection), the likelihood
39 function became harder and harder to calculate.

40 Approximate Bayesian Computation (ABC) methods (see Marin et al., 2012) dramatically extended the
41 complexity limits of the models under which inference was possible. ABC bypasses the complex task of
42 evaluating the likelihood function by combining two approximations making the problem computationally
43 tractable: (i) observed data are reduced to lower-dimensional quantities (the so-called summary statistics),
44 (ii) the inference is tolerant to small distortions of the observed summary statistics (see Blum et al. 2013 for
45 more formal explanations). This makes possible for ABC procedures to estimate posterior densities of the
46 parameters by simulating data under the model while exploring the parameter space conditionally to a prior
47 distribution, and accepting only the values of the parameters for which simulated data are close enough to
48 the observations. Despite its apparent ease, ABC methods present important methodological pitfalls (for
49 example the choice of the dimensional reduction function), but many studies have paved the way for the
50 non-statisticians (see Bertorelle et al., 2010) for an excellent methodological guide), and ABC became very
51 popular in Ecology and Evolution (Beaumont, 2010; Csilléry et al., 2010).

52 The popularity of ABC methods encouraged the development of more complex coalescence-based sim-

53 ulation computer programs, and their authors put tremendous efforts in successfully delivering novative,
54 usefull and user-friendly products to the community of population geneticists. SPLATCHE (Currat et al.,
55 2004) simulates coalescents based on complex demographic simulations run in a spatially explicit landscape,
56 incorporating landscape heterogeneity. Various versions of SPLATCHE largely fostered the rapid expansion
57 of the so-coined iDDC modeling approach (integrated distributional, demographic and coalescent model-
58 ing, He et al. 2013a). iDDC uses Approximate Bayesian Computation with spatially explicit demographic
59 simulation model (possibly integrating landscape heterogeneity) to estimate quantities of interest such as
60 populations growth rate or dispersal law parameters (Lacey Knowles and Alvarado-Serrano, 2010; Estoup
61 et al., 2010; Massatti and Knowles, 2016). DIY ABC (Cornuet et al., 2014) is an open-source program that
62 provides the ability to conduct inference under a wide range of complex biological scenarios combining an
63 arbitrary number of admixture, divergence or demographic change events. It offers very strong ABC sup-
64 port and an intuitive Graphic User Interface (GUI). IBDsim (Leblois et al., 2009) is an open-source program
65 for simulating genetic variation under isolation by distance, and provides much flexibility in the choice of
66 dispersal kernels. MSMS (Ewing and Hermisson, 2010) puts emphasis on incorporating selection and propos-
67 ing extensible design. These programs, and others, have provided invaluable support to the non-developer
68 communities for a wide range of applications and studies.

69 Paradoxically, these programs collectively failed to help the software developers community to write new
70 programs, mainly due to very low rate of reusability in their source code. Because they put emphasis on
71 the non-developer user, they act as rigid black boxes taking an input, processing it in some way configured
72 by some form of User Interface (*e.g.* command line or GUI) and delivering the output. Indeed they can not
73 be used if the underlying theoretical model does not belong to their predefined set of possible options (*e.g.*
74 SPLATCHE does not allow to change the dispersal kernel or the local growth model) or if their computational
75 solution does not answer to the question (*e.g.* if they write standard genetic summary statistics in output
76 files when we would need to analyze genealogical properties). This current state of the art does not scale
77 with the virtually infinite number of arbitrarily complex evolutionary or demographic models and the ever-
78 growing number of statistical methods variants. We need standard, general, reusable tools for helping us
79 to quickly build programs that can solve new problems. As written in Stroustrup (2003): “The key to fast
80 development, correctness, efficiency, and maintainability is to use a suitable level of abstraction supported by
81 good libraries“. Reusable code such as library’s relies on abstractions, syntactic constructions often opposed
82 to performance. However, when it comes to ABC and massive simulations, performances become critical.
83 C++ offers the template mechanism, a key feature allowing to build very high levels of abstraction without

84 loss of efficiency, so we do not have to choose between reusability and performance.

85 As a first attempt to offer reusable components to the coalescence software community, we present here
86 Quetzal, an open-source C++ template library. Quetzal offers powerful abstractions for building coalescence-
87 based simulation programs. It contains several independent modules, each directed towards a general sim-
88 ulation purpose (demography, geography, coalescence, genetics, ABC ...) in which are located the files
89 containing the sources of generic components. The extensive documentation and the wiki (Becheler, 2017),
90 both available on the github project, ease to pick and combine the desired functionalities and the template
91 mechanism allows to adapt it to a new problem with minimal recoding (if no recoding at all). We insist on
92 the possibility to extend the behaviors of the Quetzal algorithms with very few new lines of code. The high
93 level of abstraction in Quetzal allows its generic components to be used to write expressive and maintainable
94 code with appreciable terseness and efficiency. Their genericity make them applicable to a large range of
95 programs and the user can change the design of its application without major changes in the code (what
96 typically arises when changing a finally-not-so-minor detail in the theoretical model). Each documented
97 functionality comes with a small demonstration program and its output, providing valuable and intuitive
98 insights on the way to manipulate the component. The library design insists heavily on modularity, ex-
99 tensibility and efficiency, and is intended to respect the standards of the Standard Template Library with
100 STL-like algorithms and interfaces. Template rules make the library header-only.

101 This first version focuses on coalescence-based environmental demogenetic models. Consequently, Quetzal
102 first features are designed to bring efficiency and flexibility in defining demographic quantities as functions
103 of space and time of landscape heterogeneity, to couple these quantities to the coalescence process, and to
104 use ABC methodology to conduct inference.

105 First we present a simple mathematical ecological model for estimating ecological features of a species
106 from a genetic sample with the ABC procedure. This model is purposely general, as it will serve as an
107 illustration to facilitate the understanding of Quetzal functionalities by enforcing its genericity, but it is still
108 of strong biological interest as it relaxes several constraints that were previously made in the literature. Of
109 course Quetzal is flexible and its use is not limited to this model as other features can be added or removed.
110 Then we present a core feature of Quetzal for the coalescence, the abstraction of the ancestry relationship
111 between a child gene copy and its parent, and we point out its importance to use and extend the library.
112 Lastly, we give an overview of Quetzal functionalities and concepts and we apply them in a demonstration
113 program implementing a fully-specified version of the previous general theoretical model.

114 **3 Ecological and mathematical demogenetic model**

115 **3.1 Motivations**

116 Let be a spatial sample S of n haploid individuals that have been sampled at time t_S across the landscape and
117 that have been genotyped at a microsatellite locus, and a dataset giving the values of environmental quantities
118 across the same landscape. From these data, we want to infer ecological properties of the species such as
119 the *niche functions* (defined here as the functions relating the environmental quantities to demographic
120 quantities, for example the local growth rate) and the dispersal function, so we need a statistical model to
121 link observed data and processes to infer (see Figure 1). We present here a description of a general bayesian
122 model for estimating these functions using an ABC framework (see Figure 2): a demographic history is
123 simulated forward in time conditionally to the features of the heterogeneous landscape, then the genealogy
124 of the sampled gene copies is simulated backward in time conditionally to the demography. The uncertainty
125 on the parameters $\theta \in \mathbb{R}^p$ to estimate is defined by the prior distribution Π from which a parameter θ is
126 sampled at each simulation.

127 **3.2 Geography**

Let consider a given set of demes \mathbb{X} (typically reduced to the geographic coordinates of their centroid).
The environment E is defined by i known ecological quantities which are functions of space and time, typi-
cally climate layers from the WorldClim global climate database (www.worldclim.org), or a niche suitability
dataset estimated from an external niche modeling step (He et al., 2013a).

$$\begin{aligned} E & : \mathbb{X} \times \mathbb{N} \mapsto \mathbb{R}^i \\ (x, t) & \mapsto (E_i(x, t))_{i \in \mathbb{I}} . \end{aligned}$$

128 **3.3 Demography**

The demographic simulation process goes from time t_0 to t_S and iteratively constructs the function N giving
the number of individuals in deme $x \in \mathbb{X}$ at time t :

$$\begin{aligned} N & : \mathbb{X} \times \mathbb{N} \mapsto \mathbb{N} \\ (x, t) & \mapsto N(x, t) . \end{aligned}$$

N is initialized by setting $N(\cdot, t_0)$ the initial distribution of individuals across demes at the first time t_0 . Typically for a biological invasion, this is restricted to the introduction site(s) with the number of introduced individuals (Estoup et al., 2010). For endemic species, paleoclimatic distribution can be considered as starting point. The number of descendants \tilde{N}_x^t in each deme is sampled in a distribution conditionally to a function of the the local density of parents, for example $\tilde{N}_x^t \sim \text{Poisson}(g(x, t))$, where g can be for example a discrete version of the logistic growth as in Currat et al. (2004).

$$g : \begin{cases} \mathbb{X} \times \mathbb{N} & \mapsto \mathbb{R}^+ \\ (x, t) & \mapsto \frac{N_x^t \times (1+r(x, t))}{1 + \frac{r(x, t) \times N_x^t}{K(x, t)}} \end{cases} .$$

The r (respectively k) term is the growth rate (respectively the carrying capacity), defined as a function of the environmental quantities with parameter θ :

$$K : \begin{cases} \mathbb{X} \times \mathbb{N} & \mapsto \mathbb{R}_+ \\ (x, t) & \mapsto f_K^\theta(E(x, t)) \end{cases} ,$$

$$r : \begin{cases} \mathbb{X} & \mapsto \mathbb{R} \\ (x, t) & \mapsto f_r^\theta(E(x, t)) \end{cases} .$$

Non-overlapping generations are considered (the parents die just after reproduction). The children dispersal is done by sampling their destination in a multinomial law, that defines $\Phi_{x,y}^t$ the number of individuals going from x to y at time t :

$$(\Phi_{x,y}^t)_{y \in \mathbb{X}} \sim \mathcal{M}(\tilde{N}_x^t, (m_{xy})_y) .$$

The term $(m_{xy})_y$ denotes the parameters of the multinomial law, giving for an individual in x its probability to go to y . These probabilities are given by the dispersal law with parameter θ :

$$m : \begin{cases} \mathbb{X}^2 & \mapsto \mathbb{R}_+ \\ (x, y) & \mapsto m^\theta(x, y) \end{cases} .$$

After migration, the number of individuals in deme x is defined by the total number of individuals converging to x :

$$N(x, t+1) = \sum_{i \in \mathbb{X}} \Phi_{i,x}^t .$$

129 3.4 Coalescence

These quantities are used for defining the coalescence process which is defined by the following stochastic process going from t_s to t_0 : knowing that a child node c is found in $j \in \mathbb{X}$, the probability for its parent p to be in $i \in \mathbb{X}$ is :

$$P(p \in i \mid e \in j) = \frac{\Phi_{i,j}^t}{\sum_k \Phi_{k,j}^t} .$$

Knowing that the parents p_1 (p_2) of nodes c_1 (c_2) are in x at time t , the probability for the children to coalesce in the same parent is :

$$P(p_1 = p_2 \mid p_1 \in i, p_2 \in i) = 1/N_i^t .$$

130 A forest of random coalescent trees is then constructed backward in time, until t_0 is reached. Note that
131 at this point the Most Recent Common Ancestor is not necessarily found, and assuming that $t_s - t_0$ is
132 small enough no neglect mutations, the simulation can end with a collection of trees rather than a complete
133 genealogy.

134 4 Abstraction of the ancestry relationship

135 4.1 Motivations

136 When exposing the concept of coalescent trees above in the mathematical model, it has been useless to define
137 exhaustively the tree properties or the exact nature of its nodes and branches. These are details humans typ-
138 ically *abstract away*, which leads to high generalization and low intellectual overhead. However, a computer
139 program has to deal with an impressive number of details, and if these details are not carefully separated
140 from the general concerns when writing the code, it leads to poor generalization (see Alexandrescu, 2001,
141 p.xvii). Indicators of a lack of generalization are typically numerous dependencies across code, monolithic
142 classes and a high rate of code duplication. Consequences are defined by Martin (2000) and Alexandrescu
143 (see 2001, p.5) as being *rigidity* (the software is difficult to change), *fragility* (the software breaks at several
144 points after a small change), *immobility* (the software can not be reused in another context, so it is entirely
145 rewritten), gruesome intellectual overhead and poor performances. Since the devil is in the details, a natural
146 solution is to write code in terms of general *abstractions* rather than in terms of *implementation details*
147 (Dependency Inversion Principle, Martin 2002), so the designed generic components can be reused in various
148 contexts.

149 4.2 Object-oriented paradigm

150 In C++, the genericity of the implementation can be realized by using inheritance and dynamic binding
151 enable by the `virtual` keyword (Object Oriented Programming, OOP). Algorithms manipulating trees
152 would then rely on an interface defined in terms of an abstract class `AbstractTree`, but will be applied on
153 instances of concrete classes that inherit from `AbstractTree` and that present the specific desired behaviors,
154 for example `TreeForStoringCoalescenceTimes` or `TreeForStoringCoalescenceDemes`. This design avoids
155 the well-known problems of a class that would expose a monolithic interface to store all possible features
156 like demes, times, mutations and others, (Single Responsibility Principle, Martin 2002) but it has a number
157 of well-known drawbacks. First, if inheritance can be very useful when the set of classes to be treated by
158 the algorithm can naturally be thought as a hierarchy of concepts, in most cases this is not the case: it
159 would result very unnatural to order them into a class hierarchy, and, importantly, it would lead to hardly
160 maintainable code design (Stroustrup, 2014). Second, it is sometimes natural to expect the algorithms
161 to work with primitive types or with STL containers, but as primitive types are not classes and as STL
162 containers are not designed for dynamic polymorphism (they have no virtual destructor), there is no hope to
163 see the object-oriented approach work with these types. Finally, the use of inheritance and virtual functions
164 can have runtime overhead because of an extra lookup in the virtual table when a virtual function is called,
165 and because virtual methods can not be inlined (Stroustrup, 2014).

166 4.3 Generic paradigm

167 All the data type manipulated by a same general algorithm do not have to be linked by the rigid hierarchical
168 relation imposed by OOP : the generic programming allows for uniform manipulation of independent types.
169 In generic programming an algorithm is not defined in terms a particular type, but in terms of a set of
170 constraints wielded on the type by the algorithm internals; this set is frequently defined as a *concept* and
171 represents an implicit interface. Thus the algorithm will work with any type fulfilling these constraints,
172 allowing for high abstraction level without loss of efficiency. Generic programming allows to implement the
173 coalescence algorithms with great generalization, making minimal assumptions on the type handled.

174 The simple task to merge two nodes uniformly at random in a sequence of nodes can be defined as follow:

- 175 1. randomly permute the k elements of the sequence
- 176 2. create of a new node P (the parent)
- 177 3. designate the first element as child of P

178 4. designate the last element as child of P

179 5. assign P to the first element.

180 6. return the $k - 1$ first elements.

181 When implementing this binary merge algorithm in Quetzal, several details need to be abstracted away
182 to preserve the generality of the algorithm definition: the nature of the nodes, the nature of the sequence,
183 of the nature of the inheritance relationship between a child node and its parent.

184 The nodes could be integers, character strings, a user-defined class or, actually, *anything else*. No con-
185 straint on this type comes from the algorithm, but various constraints can come from the sequence type used
186 to store them. The sequence could be a standard container (`std::vector`, `std::list` ...) or a user-defined
187 type. The classical way to abstract containers in C++ is passing as argument two iterators (one pointing to
188 the first element of the sequence and the other pointing to the past-the-end element) giving the range of data
189 on which the algorithm will operate. As the algorithm can not modify the external container, the reduction
190 in size caused by the merge is signaled by returning an iterator pointing to the new past-the-end element.
191 The only explicit effort the user can have to do is just to precise, conditionally to the chosen node type, what
192 is meant by “*designating node c as child of the parent node p* “. This can done by passing to the algorithm a
193 function-object taking as argument a reference on the parent and another on the child, returning the result
194 of the branching event. Or, if no function-object is given, the sum operand is used by default (thus requiring
195 the expression $c + p$ to be defined).

196 This abstraction of the ancestry relationship is expected to allow (i) to efficiently generalize the existing
197 Quetzal algorithms to an open-ended number of specific, user-defined kinds of genealogies and (ii) to give
198 guidance to the developers who need to write new generic coalescence algorithms.

199 **4.3.1 Count hanging subtrees leaves**

200 Achieving genericity is of fundamental importance to efficiently tackle a wide range of situations. For example
201 most of the current simulation softwares focus on generating genetic variation samples, because this is most
202 of the time the only information available. However, in some cases the mutational process can be negligible
203 compared to the recent genealogical process in shaping the sample configuration (see Becheler et al., in
204 preparation), so topological properties like the number of leaves of hanging subtrees (see Hein et al., 2004,
205 p.78) become the desired output. This is very unlikely that the developer of a program could ever foresee
206 this specific need. Fortunately, it does not mean one should recode everything from scratch each time a

207 new simulation behavior is needed by a new methodological advance: Quetzal allows the user to inject the
208 desired behaviors into its generic components.

209 When implementing the simulation, instead of building complex genealogical objects, then counting their
210 leaves by tree traversals algorithms, a much more efficient approach is to directly make the coalescence
211 algorithm sum the number of leaves of the hanging subtrees, updating it at each coalescence event. The
212 type of nodes is thus defined as being integers, and the sampled nodes value is set to 1. Conveniently and
213 by default, the merging algorithm will initialize the new parents to their default constructor value (which is
214 0 for integers), and define the branching event of two nodes by the sum function.

215 The following small program applies the approach by merging two nodes uniformly at random in a
216 sequence of four nodes, updating the leaves number information. It can of course be extended to much more
217 complex simulation frameworks.

```
218  
219 #include "quetzal/coalescence.h"  
220 #include <random> // std::mt19937  
221 #include <iostream> // std::cout  
222 #include <algorithm> // std::copy  
223 #include <iterator>  
224  
225 using namespace quetzal::coalescence;  
226  
227 int main(){  
228     using node_type = int;  
229     std::vector<node_type> nodes(4,1);  
230     std::mt19937 rng;  
231     auto last = binary_merge(nodes.begin(), nodes.end(), rng);  
232  
233     std::ostream_iterator<node_type> it(std::cout, "␣");  
234     std::copy(nodes.begin(), last, it);  
235     return 0;  
236 }
```

237 The output gives the number of leaves of each hanging subtree after one generation of coalescence:

238 2 1 1

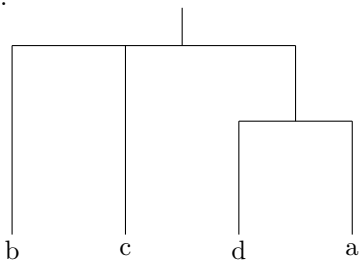
239 4.3.2 Construct a Newick tree format

240 Code with a suitable level of abstraction allows to readapt old code to new problems with ease. Studying
241 genealogies topological properties can be the main statistical focus, but visualizing genealogies is still the
242 most instinctive way to shed light on some properties, to assert correctness of algorithms generating them,
243 or to present results. However, when it comes to data visualization, C++ is not the most suited platform.
244 Many tree visualizer use a Newick tree format (see Olsen, 1990) as an input for nice plot rendering. The
245 implementation is straightforward when using Quetzal abstractions: the type of the nodes is now a character
246 string, the parent node is by default constructed as an empty string, and the branching event is defined
247 as a formating function taking the parent node p and the child node c as argument to build the Newick
248 format character string piece by piece. There are very few lines to change in the code to entirely redefine
249 the meaning of a coalescence event:

```
250  
251 #include "quetzal/coalescence.h"  
252 #include <random> // std::mt19937  
253 #include <iostream> // std::cout  
254 #include <algorithm> // std::copy  
255 #include <iterator>  
256 #include <string>  
257 using namespace quetzal::coalescence;  
258  
259 int main(){  
260     using node_type = std::string;  
261  
262     std::vector<node_type> nodes = {"a","b","c","d"};  
263     std::mt19937 rng;  
264  
265     auto branch = [](auto p, auto c){  
266         if(p.size() == 0)
```

```
267     return "(" + c;  
268     else  
269     return p + "," + c + "));  
270 };  
271  
272 auto first = nodes.begin();  
273 auto last = nodes.end();  
274  
275 while(distance(first, last) > 1){  
276     last = binary_merge(first, last, rng, branch);  
277 }  
278  
279 std::ostream_iterator<node_type> it(std::cout, "\n");  
280 std::copy(nodes.begin(), last, it);  
281 return 0;  
282 }
```

283 The output will give the Newick format character string ((d,a),b,c) representing the following coalescent
284 tree.



285
286 The output can be exported for example on an online tree viewer such as iTol (Letunic and Bork, 2006).
287 Quetzal is not restricted to this simple example and any arbitrarily more complex formatting functions can
288 be considered, for example to represent branches length or nodes position in a landscape.

289 5 Quetzal components for simulation

290 The manipulation of the genealogies is the most fundamental aspect of all coalescence-based application
291 programs, so the abstraction of the ancestry relationship is expected to be always useful, and the algorithms

292 written in terms of this abstraction are expected to be highly reusable. However, an open-ended number of
293 generative model variants can be considered: we present here a number of components that are most likely
294 to be necessary when implementing them. Note that these components are intended to be independent.
295 For example, if a demographic simulation is usually run after reading some environmental quantities in
296 a geographic file, this does not have to be the case. Indeed any user-defined set of coordinates can be
297 used to represent the demic structure, and environmental quantities can for example be represented by any
298 mathematical function of the geographic space. Accordingly, the type of the geographic coordinates used in
299 the *geography* module does not pervade the other modules.

300 5.1 Discrete landscape construction

301 In the *geography* module, Quetzal uses the Geospatial Data Abstraction Library (GDAL Development Team,
302 2017) to read grids of ecological data through the instantiation of a `DiscreteLandscape` object. In this class,
303 the demes are represented by the grid cells and identified by the geographic coordinate of their centroid. The
304 class allows to retrieve the set of demes centroid geographic coordinates (so it can be used to represent the
305 demic structure to run spatially explicit simulations), to reproject a set of sampled coordinates to the nearest
306 centroid (so compatibility is ensured between a spatial sample and the geographic support), or to deliver
307 lightweight function-objects that give access to the underlying ecological quantities and that are susceptible to
308 be coupled to the demographic model by composing them into arbitrarily complex mathematical expressions
309 of space and time. The `GeographicCoordinates` class allows secured manipulations of longitude and latitude
310 coordinates, and computation of great circle distances (often useful to define dispersal kernels).

311 5.2 Demographic variables definition

312 In the *demography* module, Quetzal defines two class templates (`PopulationSize` and `PopulationFlux`) to
313 construct and consult (N_x^t) and (Φ_{ij}^t) , providing expressive interface for secured and intuitive manipulation.
314 Both class templates do not depend on *geography* module as they are templated on the type of the locations
315 (the demes) and on the type of values that are stored. It makes possible to use any geographic coordinate
316 system (for example longitude/latitude using the `geography::GeographicCoordinates`) to spatialize the
317 model. Any arithmetic type can be chosen to define the set in which (N_x^t) and (Φ_{ij}^t) take value (typically \mathbf{N}
318 or $\mathbf{R}+$). Indeed, the type of the stored time values is not necessarily an integer but can be a more complex
319 date type.

320 5.3 Compile-time functions composition

321 Because growth patterns are species-specific features, the expression of N_x^{t+1} is typically user-defined and
322 hence can be any arbitrarily complex function, for example constant values or discrete version of logistic
323 growth model (Currat et al., 2004). Quetzal offers tremendous facilities to build these functions, composing
324 function-objects into an expression that can be efficiently passed around. To this purpose, Quetzal integrates
325 *expressive* (Marques, 2017), a library making use of metaprogramming techniques to enable compile-time
326 optimization of function composition with high expressiveness.

327 Consider the discrete logistic growth version (see section 3) to define N_x^{t+1} and let us pretend there is
328 strong biological motivations to want the growth rate r to be constant ($r = 3$) over space and time, and
329 the carrying capacity K to be the mean of heterogeneous environmental quantities E_1 and E_2 , $K = \frac{E_1+E_2}{2}$.
330 As C++ has strong static typing, it is impossible to directly sum constants (literals) with functions, as it
331 would be possible under others languages. So the first step is to transform the constants as functions with
332 the same definition space as the other functions we want to combine:

```
333 literal_factory<space_type, time_type> lit;  
334 auto r = lit(3);
```

335 Here r is now callable with time and space arguments, and can be composed with other *expressive* callable
336 objects. Assuming that E_1 and E_2 are function-objects callable with time and space arguments (for exam-
337 ple the function-objects delivered by the `DiscreteLandscape` class), the function `use` allows *expressive* to
338 manipulate the expressions and gives them an enriched mathematical interface, so the addition or division
339 operator can be applied on them:

```
340 auto K = (use(E_1)+use(E_2))/lit(2);
```

341 And finally, assuming that N is a function-object callable with space and time arguments, the whole
342 growth expression can be built:

```
343 auto g = (use(N)*(lit(1)+r)/(lit(1)+((r * use(N))/K)));
```

344 This expression can be captured in a lambda expression to simulate the number of gene copies after repro-
345 duction in deme x at time t :

```
346 auto sim_growth = [g](auto x, auto t, auto& gen){  
347     poisson_distribution<N_type> dis(g(x,t));  
348     return dis(gen);
```

349 }

350 This object can be passed around conveniently for further invocation with space and time arguments:

```
351 for(auto t : times){
352     for(auto x : space){
353         // ...
354         auto N_tilde=sim_growth(x,t,gen);
355         // ...
356     }
357 }
```

358 This last code snippet illustrates an important benefit of *expressive*: to allow the separation of concerns when
359 writing the application code. In other terms, the details of the logistic growth expression are not intricate
360 with the code where the expression is invoked, what would result in an obfuscated and hardly maintainable
361 code. Moreover, as the expression is known at compile-time, it is a perfect candidate for all compile-time
362 optimizations done by the compiler such as inlining: as the compiler knows exactly which functions will be
363 called when g is called, he should be able to replace a function call directly by the body of the function,
364 potentially leading to extremely efficient code with very few indirections.

365 5.4 Dispersal patterns

366 In the *random* module, the `TransitionKernel` class is an implementation of a markovian kernel for sampling
367 the next state x_1 of a markovian process conditionally to the present state x_0 . It can be used in the dispersal
368 context (in that case the states type will represent demes coordinates, for example `geography::GeographicCoordinates`).
369 The underlying markovian probability distributions associated with each present state do not need to have
370 the same arrival space, and they are built only if needed by the simulation context. The weights are com-
371 puted with an arbitrary mathematical function conditionally to the present state (for example when only
372 geographic distance affects dispersal) or to the present state and time (for example when environmental
373 spatio-temporal heterogeneity affects dispersal).

374 5.5 Coalescence features

375 For coalescence under the Wright-Fisher model, a binary merge algorithm is proposed to be used in the
376 simulation contexts where the sample size is small relative to the population size. A simultaneous multiple

377 merge algorithm can be used when this approximation does not hold.

378 The benefit of abstracting the inheritance relationship when simulating the genealogical graph was pre-
379 sented above along with two examples showing that an explicit representation of the coalescent was not
380 necessarily desirable. However in many standard cases, it is needed, for example to save arbitrary infor-
381 mation from the simulation context (times of coalescence events) and access them later for updating some
382 quantities while descending the genealogy (for example apposing mutations with a probability conditional
383 to the time spent between two nodes). For these cases, the `Tree` class template allows to construct such
384 object, encapsulating an arbitrary user-defined data field into each node, defining the inheritance relationship
385 between a parent node and a child node in a secured way, proposing topological manipulation operations
386 and tree traversal algorithms.

387 The `Forest` class template is designed to ease the manipulation of spatial collections of trees (of arbitrary
388 type) when using spatially explicit coalescence simulation.

389 **6 Quetzal components for inference**

390 The Quetzal *abc* module provides abstractions allowing to embed efficiently an open-ended range of simula-
391 tion models into an ABC framework. To this purpose, an ABC object associates a simulation model and the
392 prior distribution of its (possibly multidimensional) parameter to conduct inference. We present here key
393 elements of the *abc* module, notably the concept of *GenerativeModel* used to abstract out the model-specific
394 details.

395 As ABC-based inference on spatial coalescents involves complex functions for dimensional reduction and
396 distance computation that are far beyond the scope of this article, the ABC inference examples will be
397 presented with a toy generative model (the poisson distribution), a toy dimensional reduction function η
398 (identity) and a toy distance ρ (absolute value of the difference).

399 Then we step away from the toy-model and propose a concrete example of a class satisfying *Generative-*
400 *Model* and implementing a fully-specified version of the general theoretical model of coalescence presented in
401 section 3. This example will make use of Quetzal components to illustrate how to build original coalescence
402 simulations objects with ABC-compatible interface.

403 6.1 Features

404 6.1.1 The *GenerativeModel* concept

405 To achieve genericity and propose clear, standard and uniform ways to manipulate models and parameters,
406 specific simulation models are abstracted to the concept of *GenerativeModel*, a Quetzal C++ concept that
407 has been designed to be a generalization of the standard C++ *RandomNumberDistribution*. It notably
408 generalizes the type of the result that is no more restricted to be an arithmetic type, so more complex type
409 values (coalescents, genetic data, or summary statistics) can be generated. Furthermore, the returned values
410 are not necessarily generated from a simple probability density function or a discrete probability distribution,
411 as generally in ABC a complex stochastic simulation function is involved. The list of all requirements can
412 be found in the documentation. Any model object whose type D satisfies *GenerativeModel* and any prior
413 object able to randomly produce an object of type `D::param_type` can be used to build an ABC object.
414 Consequently, all the STL random number distributions are compatible with the *abc* module, which is very
415 convenient for testing and demonstration purposes:

```
416 using model_type = poisson_distribution<>;  
417 uniform_real_distribution<double> prior(1.,100.);  
418 model_type model;  
419 auto abc = make_ABC(model, prior);
```

420 Here an ABC object is constructed by associating the STL poisson distribution with a prior on its
421 parameter, the STL uniform distribution, for sampling parameters in $[0,100]$.

422 6.1.2 Prior predictive distribution sampling

423 The generation of the reference table is done by sampling n results in the prior predictive distribution.

```
424 mt19937 g;  
425 auto n = 1000000;  
426 auto table = abc.sample_prior_predictive_distribution(n,g);
```

427 The generated `ReferenceTable` object can compute other table objects. Considering a function-object
428 representing the summary statistics function η , the raw data table can produce a second `ReferenceTable`
429 object associating the parameter value to generated summary statistics.

```
430 auto eta = [](auto x){return x;};
```

```
431 auto sumstats = table.compute(eta);
```

432 Generated data can be accessed, for example to be used as pseudo-observed data in ABC validation
433 methodology:

```
434 auto pod_value = sumstats.begin()->value();
```

```
435 auto pod_param = sumstats.begin()->param();
```

436 Then, considering a function object representing ρ , the distance function between observed and simulated
437 dataset,

```
438 auto rho = [](auto obs, auto sim){return abs(obs - sim);};
```

```
439 auto distances = sumstats.compute_distance_to(pod, rho);
```

440 Finally the syntax of the various interfaces make it intuitive to design a quick rejection algorithm, sending
441 to output only the parameter values for which the generated summary statistics was less than a threshold:

```
442 double threshold = 2.0;
```

```
443 for(auto const&it : distances){
```

```
444     if(it.value() <= threshold){
```

```
445         cout << it.param().lambda() << endl;
```

```
446     }
```

```
447 }
```

448 6.1.3 Rejection samplers

449 The simplest samplers is the Rubin rejection sampler (Rubin, 1984). It accepts a parameter value only if
450 the generated data is strictly equal to the observed data (the data type has to be *EqualityComparable*, *i.e.*
451 having a built-in or a user-defined comparison operator `operator==`).

452 The implementation of the Pritchard rejection sampler (Pritchard et al., 1999) generalizes the dimensional
453 reduction function (that traditionally computes summary statistics) and the distance function (that evaluates
454 the distance between observation and simulation). Therefore, any object-function can be used to transform
455 data into summary statistics and any type of distance can be used.

456 More complex sampling algorithms like MCMC-ABC (Marjoram et al., 2003), SMC-ABC (Del Moral
457 et al., 2006), or PMC-ABC (Beaumont et al., 2009) are yet not implemented, but we do not expect that it
458 will hinder Quetzal reliability. Indeed these algorithms are known to be challenging to calibrate (Marin et al.,

2012), while embedding the simulation model and the inference framework in the same C++ application code has the benefit to make all the type information available for the compiler, decreasing the computational cost, so the generation of the reference table alone is expected to be useful for a wide range of situations. Furthermore, we expect that if more sophisticated versions of algorithms are needed, the Quetzal existing concepts will greatly ease their implementation.

6.2 Implementing a custom generative model

We present here how to construct a class `ExampleModel` that meets the requirements of the *GenerativeModel*. The main general ideas are highlighted here and the program can be found in the supplementary material.

We consider a landscape reduced to two demes A and B . At time t_0 , $N_0 = 10$ haploid individuals were introduced in deme A . The local growth rate and the local carrying capacity K are assumed to be constant across the landscape. The growth rate is known ($r = 100$) while the local carrying capacity K is unknown and assumed to belong to $[1, 500]$. The aim of the program is to estimate this value starting with a uniform prior distribution on $[1, 500]$. For each individual there is a probability $m = 0.1$ to migrate to the other deme. After $g = 10$ generations, $n = 30$ individuals were sampled in B and genotyped at one locus. We assume that each introduced individual had different allelic states and that mutational process is negligible. Under these hypotheses, the observed clustering of the data is only shaped by the genealogical process, so we can reject all simulated coalescent forests that do not clusterize the dataset into as many subsets of same cardinality than in the observed clustering. Consequently, we just need to construct the vector of the hanging subtrees leaves count (a way to do it efficiently was presented above) and to compare it to the observed vector of clusters size. We accept the parameter used for the simulation only if the two vectors are equals. For a demonstration purpose, we construct a reference table by sampling 5×10^5 simulated data into the prior predictive distribution, and we generate 100 pseudo-observed data under the parameter $K = 50$ for validation. Note here that if the pseudo-observed data do not contain 30 individuals in deme B , no posterior will be estimated. The prior and posterior distributions are shown in Figure 3.

6.2.1 ABC-compatible interface

Declaring the following interface is sufficient to capt the generality of all possible generative models and enable the `ExampleModel` class to interact with an ABC object:

```
class ExampleModel{
public:
```

```
488     using param_type = Param ;
489     using value_type = ... ;
490
491     template<typename Generator>
492     value_type operator()(Generator& gen, param_type const& p) const;
493 };
```

494 The member type `value_type` describes the type of value generated by the model. The member type
495 `param_type` encapsulates the details of θ , the multidimensional parameter to estimate.

496 6.2.2 Encapsulating θ

497 The point here is to hide useless details (such as dimensionality) from ABC procedures while allowing
498 the user to have control over its implementation. We warn against using vectors or arrays to represent
499 multidimensional parameters: it would impose all dimensions to be of same type, and their index-based
500 value access interface would later favor confusion between dimensions. Instead, we suggest to follow the STL
501 standards and to implement user-defined small classes, with expressive getter/setter syntax. Here we show
502 an extract of the `Param` class, where the member `k` represents K , the carrying capacity of each deme in the
503 landscape. Other dimensions are not given here, but can be set to constants for better code maintainability,
504 as found in the supplementary material.

```
505 class Param{
506 private:
507     double K;
508 public:
509     double K() const; // getter
510     void K(double); // setter
511     // ... other dimensions
512 };
```

513 6.2.3 Constructing the prior

514 Instances of this parameter class can be created in a prior distribution *i.e* a function that can be called with
515 a random generator and that randomly produces a `param_type` object, manipulating the `Param` object *via*

516 its interface to set its dimension values:

```
517 auto prior = [](auto& gen){
518     ExampleModel::param_type params;
519     params.k(std::uniform_int_distribution<>(1,500)(gen));
520     params.r(100);
521     params.m(0.1);
522     return params;
523 };
```

524 More guidance in the design of this second-order function can be found in the project wiki. This function-
525 object, representing the parameter joint distribution, will be passed to the ABC object, that will use it to
526 generate random parameters and pass them to the model `ExampleModel::operator()` member function to
527 generate random `value_type` objects. The model details lay in the definition of `ExampleModel::operator()`
528 member function, and a possible implementation is proposed in the supplementary material.

529 7 Acknowledgements

530 We thank Ambre Marques who importantly contributed to the present Quetzal state by taking on her
531 free-time to provide advice on generic paradigm and design issues, and to develop the *expressive* library.

532 We thank Florence Jornod who participated as an intern.

533 Arnaud Becheler was funded by a multidisciplinary project founded by the French Government (LabEx
534 BASC, ANR-11-LABX-0034) that aims to provide new knowledge regarding the drivers of species distribution
535 and to design innovative guidelines toward sustainable land management.

536 This work was partially supported by the Chair "Modélisation Mathématique et Biodiversité" of VEOLIA-
537 Ecole Polytechnique-MNHN-F.X., by the Mission for Interdisciplinarity at CNRS and by the Institute for
538 the Diversity, Ecology and Evolution of the Living World.

539 References

540 Alexandrescu, A. (2001). *Modern C++ design: generic programming and design patterns applied*. Addison-
541 Wesley.

- 542 Beaumont, M. A. (2010). Approximate Bayesian Computation in Evolution and Ecology. *Annual Review of*
543 *Ecology, Evolution, and Systematics*, 41(1):379–406.
- 544 Beaumont, M. A., Cornuet, J.-M., Marin, J.-M., and Robert, C. P. (2009). Adaptive approximate Bayesian
545 computation. *Biometrika*, 96(4):983–990.
- 546 Becheler, A. (2017). Quetzal. <https://github.com/Becheler/quetzal>. [Online; accessed 28-September-
547 2017].
- 548 Beerli, P. and Felsenstein, J. (1999). Maximum-Likelihood Estimation of Migration Rates and Effective
549 Population Numbers in Two Populations Using a Coalescent Approach. *Genetics*, 152(2):763–773.
- 550 Bertorelle, G., Benazzo, A., and Mona, S. (2010). ABC as a flexible framework to estimate demography over
551 space and time: some cons, many pros: the abc revolution in nine steps. *Molecular Ecology*, 19(13):2609–
552 2625.
- 553 Blum, M. G. B., Nunes, M. A., Prangle, D., and Sisson, S. A. (2013). A Comparative Review of Dimension
554 Reduction Methods in Approximate Bayesian Computation. *Statistical Science*, 28(2):189–208.
- 555 Cornuet, J.-M., Pudlo, P., Veyssier, J., Dehne-Garcia, A., Gautier, M., Leblois, R., Marin, J.-M., and
556 Estoup, A. (2014). DIYABC v2.0: a software to make approximate Bayesian computation inferences
557 about population history using single nucleotide polymorphism, DNA sequence and microsatellite data.
558 *Bioinformatics*, 30(8):1187–1189.
- 559 Csilléry, K., Blum, M. G., Gaggiotti, O. E., and François, O. (2010). Approximate Bayesian Computation
560 (ABC) in practice. *Trends in Ecology & Evolution*, 25(7):410–418.
- 561 Currat, M., Ray, N., and Excoffier, L. (2004). splatche: a program to simulate genetic diversity taking into
562 account environmental heterogeneity: PROGRAM NOTE. *Molecular Ecology Notes*, 4(1):139–142.
- 563 Del Moral, P., Doucet, A., and Jasra, A. (2006). Sequential monte carlo samplers. *Journal of the Royal*
564 *Statistical Society: Series B (Statistical Methodology)*, 68(3):411–436.
- 565 Estoup, A., Baird, S. J., Ray, N., Currat, M., Cornuet, J.-M., Santos, F., Beaumont, M. A., and Excoffier,
566 L. (2010). Combining genetic, historical and geographical data to reconstruct the dynamics of bioinva-
567 sions: application to the cane toad *Bufo marinus*: reconstructing bioinvasion dynamics. *Molecular Ecology*
568 *Resources*, 10(5):886–901.

- 569 Ewing, G. and Hermisson, J. (2010). MSMS: a coalescent simulation program including recombination,
570 demographic structure and selection at a single locus. *Bioinformatics*, 26(16):2064–2065.
- 571 Excoffier, L., Foll, M., and Petit, R. J. (2009). Genetic Consequences of Range Expansions. *Annual Review*
572 *of Ecology, Evolution, and Systematics*, 40(1):481–501.
- 573 GDAL Development Team (2017). *GDAL - Geospatial Data Abstraction Library, Version 2.2.1*. Open Source
574 Geospatial Foundation. [Online; accessed 28-September-2017].
- 575 He, Q., Edwards, D. L., and Knowles, L. L. (2013a). Integrative testing of how environments from the past
576 to the present shape genetic structure across landscapes. *Evolution*, 67(12):3386–3402.
- 577 He, Q., Edwards, D. L., and Knowles, L. L. (2013b). Integrative testing of how environments from the past
578 to the present shape genetic structure across landscapes. *Evolution*, 67(12):3386–3402.
- 579 Hein, J., Schierup, M., and Wiuf, C. (2004). *Gene Genealogies, Variation and Evolution: A primer in*
580 *coalescent theory*. Oxford University Press, USA.
- 581 Kuhner, M. K., Yamato, J., and Felsenstein, J. (2000). Maximum likelihood estimation of recombination
582 rates from population data. *Genetics*, 156(3):1393–1401.
- 583 Lacey Knowles, L. and Alvarado-Serrano, D. F. (2010). Exploring the population genetic consequences of the
584 colonization process with spatio-temporally explicit models: insights from coupled ecological, demographic
585 and genetic models in montane grasshoppers: genetic consequence of distribution shifts. *Molecular Ecology*,
586 19(17):3727–3745.
- 587 Leblois, R., Estoup, A., and Rousset, F. (2009). IBDSim: a computer program to simulate genotypic data
588 under isolation by distance. *Molecular Ecology Resources*, 9(1):107–109.
- 589 Letunic, I. and Bork, P. (2006). Interactive Tree Of Life (iTOL): an online tool for phylogenetic tree display
590 and annotation. *Bioinformatics*, 23(1):127–128.
- 591 Marin, J.-M., Pudlo, P., Robert, C. P., and Ryder, R. J. (2012). Approximate Bayesian computational
592 methods. *Statistics and Computing*, 22(6):1167–1180.
- 593 Marjoram, P., Molitor, J., Plagnol, V., and Tavaré, S. (2003). Markov chain Monte Carlo without likelihoods.
594 *Proceedings of the National Academy of Sciences*, 100(26):15324–15328.

- 595 Marques, A. (2017). expressive. <https://github.com/ambre-m/expressive>. [Online; accessed 28-
596 September-2017].
- 597 Martin, R. C. (2000). Design principles and design patterns. *Object Mentor*, 1(34).
- 598 Martin, R. C. (2002). *Agile software development: principles, patterns, and practices*. Prentice Hall.
- 599 Massatti, R. and Knowles, L. L. (2016). Contrasting support for alternative models of genomic variation
600 based on microhabitat preference: species-specific effects of climate change in alpine sedges. *Molecular*
601 *Ecology*, 25(16):3974–3986.
- 602 Nordborg, M. (2001). Coalescent theory. *Handbook of statistical genetics*.
- 603 Olsen, G. (1990). Gary Olsen's Interpretation of the "Newick's 8:45" Tree Format Standard. http://evolution.genetics.washington.edu/phylip/newick_doc.html. [Online; accessed 28-September-
604 2017].
- 605
- 606 Pauls, S. U., Nowak, C., Bálint, M., and Pfenninger, M. (2013). The impact of global climate change on
607 genetic diversity within populations and species. *Molecular Ecology*, 22(4):925–946.
- 608 Pritchard, J. K., Seielstad, M. T., Perez-Lezaun, A., and Feldman, M. W. (1999). Population growth of
609 human Y chromosomes: a study of Y chromosome microsatellites. *Molecular Biology and Evolution*,
610 16(12):1791–1798.
- 611 Rubin, D. (1984). Bayesianly justifiable and relevant frequency calculations for the applied statistician.
612 *Annals of statistics*, 12(4):1151–1172.
- 613 Stroustrup, B. (2003). Abstraction, libraries, and efficiency in C+. [http://www.stroustrup.com/
614 abstraction.pdf](http://www.stroustrup.com/abstraction.pdf). [Online; accessed 28-September-2017].
- 615 Stroustrup, B. (2014). Five Popular Myths about C++. <http://www.stroustrup.com/Myths-final.pdf>.
616 [Online; accessed 28-September-2017].
- 617 Wakeley, J. (2009). *Coalescent theory: an introduction*. Number 575: 519.2 WAK.

618 8 Data Accessibility

619 Quetzal source code can be found on github project (<https://github.com/Becheler/quetzal>). The README
620 file redirects towards Quetzal resources (documentation, wiki, IRC channel). This program is a free software;

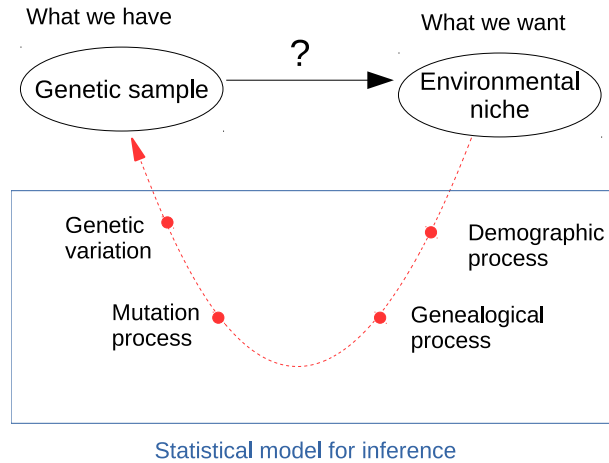


Figure 1: Inferential framework from which Quetzal stems. The red arrow represents the dependency structure under the hypothesis of neutral variation, where processes condition the observed data. The black arrow represents the inferential framework, where data allow to shed light on processes.

621 you can redistribute it and/or modify it under the terms of the GNU General Public License as published
622 by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

623 9 Authors Contribution

624 All authors participated equally in the mathematical model design. Arnaud Becheler implemented the C++
625 library. The article and the documentation of the Quetzal project were written by Arnaud Becheler in
626 cooperation with the other authors.

627 10 Tables and Figures

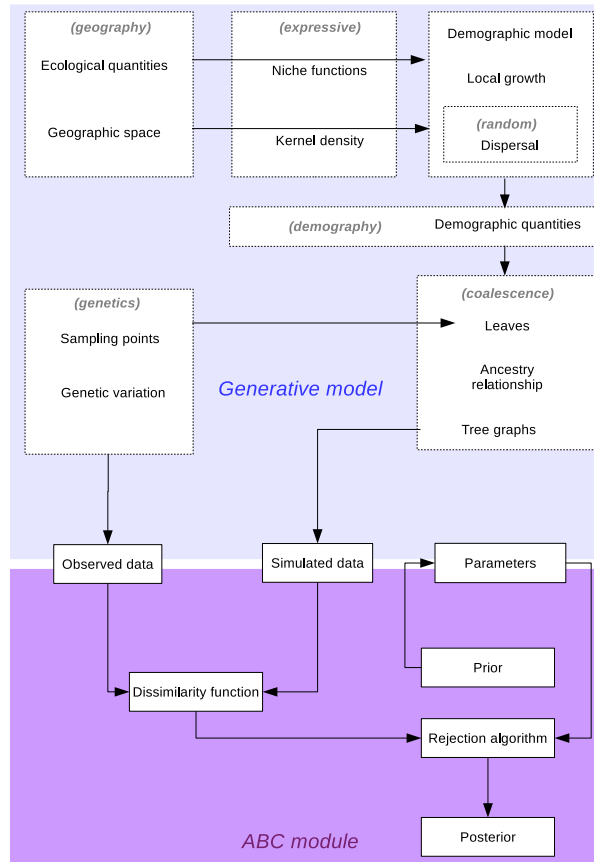


Figure 2: Flow chart illustrating the information flows between components of a general environmental demogenetic model (see section 3). In grey parenthesis are indicated the Quetzal modules (see section 5) that allow to represent these components in the code. The way the landscape conditions the demographic processes form the main focus of a number of approaches (landscape-ABC (Estoup et al., 2010), iDDC modeling (He et al., 2013b)) in the literature, such that the inference is usually driven on the underlying niche and/or dispersal model. Inferring such ecological properties from a spatial genetic sample is made possible by using a coalescence model to link the sample to the demographic processes that shaped it. Inference is run in an ABC framework, where parameters to estimate are sampled in a distribution, allowing a dataset simulated by the generative model to be compared to the observed data by some dissimilarity function to build the posterior.

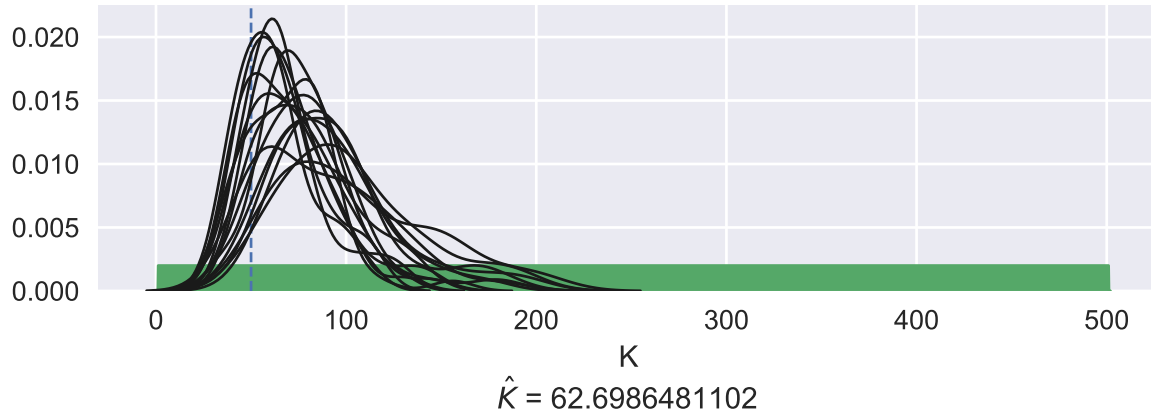


Figure 3: Posterior densities obtained by ABC inference conducted on pseudo-observed data generated under the *ExampleModel* model shown in section 6.2. True parameter value $K = 50$ is shown as the vertical dashed line. The prior distribution is shown in green. The mean of all posteriors is given as \hat{K} .