```
163         **test_components**.
164 
165         :param test_components: list of components whose presence is checked
166         :type test_components: sequence of :class:`RVComp` items
167         """
168         for test_comp in test_components:
169             if not self.contains(test_comp):
170                 return False
171         return True;
172 
173     def contains_any(self, test_components):
174         """Return True if this RV contains any of **test_components**.
175 
176         :param test_components: sequence of components whose presence is tested
177         :type test_components: sequence of :class:`RVComp` items
178         """
179         for test_comp in test_components:
180             if self.contains(test_comp):
181                 return True
182         return False
183 
184     def contained_in(self, test_components):
185         """Return True if sequence **test_components** contains all components
186         from this RV (and perhaps more).
187 
188         :param test_components: set of components whose presence is checked
189         :type test_components: sequence of :class:`RVComp` items
190         """
191         for component in self.components:
192             if component not in test_components:
193                 return False
194         return True
195 
196     def indexed_in(self, super_rv):
197         """Return index array such that this rv is indexed in **super_rv**, which
198         must be a superset of this rv. Resulting array can be used with :func:`numpy.take`
199         and :func:`numpy.put`.
200 
201         :param super_rv: returned indices apply to this rv
202         :type super_rv: :class:`RV`
203         :rtype: 1D :class:`numpy.ndarray` of ints with dimension = self.dimension
204         """
205         ret = np.empty(self.dimension, dtype=int)
206         ret_ind = 0  # current index in returned index array
207         # process each component from target rv
208         for comp in self.components:
209             # find associated component in source_rv components:
210             src_ind = 0  # index in source vector
211             for source_comp in super_rv.components:
212                 if source_comp is comp:
213                     ret[ret_ind:] = np.arange(src_ind, src_ind + comp.dimension)
214                     ret_ind += comp.dimension
215                     break;
216                 src_ind += source_comp.dimension
217             else:
218                 raise AttributeError("Cannont find component "+str(comp)+" in source_rv.components.")
219         return ret
220 
221 
222 class CPdf(object):
223     r"""Base class for all Conditional (in general) Probability Density Functions.
224 
225     When you evaluate a CPdf the result generally also depends on a condition
226     (vector) named `cond` in PyBayes. For a CPdf that is a :class:`Pdf` this is
227     not the case, the result is unconditional.
228 
229     Every CPdf takes (apart from others) 2 optional arguments to constructor:
230     **rv** (:class:`RV`) and **cond_rv** (:class:`RV`). When specified, they
231     denote that the CPdf is associated with a particular random variable (respectively
232     its condition is associated with a particular random variable); when unspecified,
233     *anonymous* random variable is assumed (exceptions exist, see :class:`ProdPdf`).
234     It is an error to pass RV whose dimension is not same as CPdf's dimension
235     (or cond dimension respectively).
236 
237     :var RV rv: associated random variable (always set in constructor, contains
238        at least one RVComp)
239     :var RV cond_rv: associated condition random variable (set in constructor to
240        potentially empty RV)
241 
242     *While you can assign different rv and cond_rv to a CPdf, you should be
243     cautious because sanity checks are only performed in constructor.*
244 
245     While entire idea of random variable associations may not be needed in simple
246     cases, it allows you to express more complicated situations. Assume the state
```