

UNGenético Manual de referencia

Generado por Doxygen 1.3.6

Abril de 2004

Índice general

1. UNGenético	1
1.1. Descripción	1
1.2. Antecedentes	2
1.3. Licencia	2
1.4. Créditos	3
2. Utilización de UNGenético 2.0	4
2.1. Sistema propuesto	4
2.2. Creación del proyecto	5
2.2.1. Creación del proyecto en consola	5
2.2.2. Creación del proyecto usando entorno gráfico	5
2.3. Definición de las propiedades del algoritmo genético	6
2.3.1. Modificación de las propiedades de la clase AlgoritmoGenetico	7
2.3.2. Modificación de los operadores del algoritmo genético	7
2.3.3. Definición de las propiedades de los individuos	9
2.3.4. Definición de la función objetivo	11
2.4. Definición del procedimiento principal en consola	12
2.5. Definición del procedimiento principal usando entorno gráfico	13
2.6. Ejecución del proyecto	15
2.7. Resumen gráfico del desempeño del algoritmo	15

ÍNDICE GENERAL

II

2.8. Utilización de UNGenético Wizard	18
2.8.1. Adición de variables	18
2.8.2. Definición de operadores genéticos	20
2.8.3. Definición de parámetros para la optimización	21
2.8.4. Creación del archivo principal	22
3. UNGenético Índice jerárquico	23
3.1. UNGenético Jerarquía de la clase	23
4. UNGenético Índice de clases	26
4.1. UNGenético Lista de componentes	26
5. UNGenético Documentación de clases	31
5.1. Referencia de la Clase AGFrame	31
5.1.1. Descripción detallada	31
5.1.2. Documentación de las enumeraciones miembro de la clase	35
5.1.3. Documentación del constructor y destructor	36
5.1.4. Documentación de las funciones miembro	36
5.1.5. Documentación de los datos miembro	38
5.2. Referencia de la Clase AGVentana	41
5.2.1. Descripción detallada	41
5.2.2. Documentación de las enumeraciones miembro de la clase	45
5.2.3. Documentación del constructor y destructor	46
5.2.4. Documentación de las funciones miembro	46
5.2.5. Documentación de los datos miembro	48
5.3. Referencia de la Clase AlgoritmoGenetico	50
5.3.1. Descripción detallada	50
5.3.2. Documentación del constructor y destructor	57
5.3.3. Documentación de las funciones miembro	57
5.3.4. Documentación de los datos miembro	64
5.4. Referencia de la Clase Arreglo	70

5.4.1. Descripción detallada	70
5.4.2. Documentación del constructor y destructor	73
5.4.3. Documentación de las funciones miembro	73
5.4.4. Documentación de los datos miembro	79
5.5. Referencia de la Clase Gen	80
5.5.1. Descripción detallada	80
5.5.2. Documentación del constructor y destructor	81
5.5.3. Documentación de las funciones miembro	81
5.6. Referencia de la Clase GenArreglo	84
5.6.1. Descripción detallada	84
5.6.2. Documentación del constructor y destructor	88
5.6.3. Documentación de las funciones miembro	89
5.6.4. Documentación de los datos miembro	94
5.7. Referencia de la Clase GenBool	96
5.7.1. Descripción detallada	96
5.7.2. Documentación del constructor y destructor	98
5.7.3. Documentación de las funciones miembro	99
5.7.4. Documentación de los datos miembro	101
5.8. Referencia de la Clase GenEntero	102
5.8.1. Descripción detallada	102
5.8.2. Documentación del constructor y destructor	104
5.8.3. Documentación de las funciones miembro	105
5.8.4. Documentación de los datos miembro	108
5.9. Referencia de la Clase GenReal	110
5.9.1. Descripción detallada	110
5.9.2. Documentación del constructor y destructor	112
5.9.3. Documentación de las funciones miembro	113
5.9.4. Documentación de los datos miembro	116
5.10. Referencia de la Clase Individuo	118

5.10.1. Descripción detallada	118
5.10.2. Documentación del constructor y destructor	121
5.10.3. Documentación de las funciones miembro	121
5.10.4. Documentación de los datos miembro	124
5.11. Referencia de la Clase OperadorAdaptacion	126
5.11.1. Descripción detallada	126
5.11.2. Documentación del constructor y destructor	127
5.11.3. Documentación de las funciones miembro	127
5.12. Referencia de la Clase OperadorAdaptacionElitismo	128
5.12.1. Descripción detallada	128
5.12.2. Documentación del constructor y destructor	129
5.12.3. Documentación de las funciones miembro	129
5.13. Referencia de la Clase OperadorAdaptacionNumIndividuos	130
5.13.1. Descripción detallada	130
5.13.2. Documentación del constructor y destructor	131
5.13.3. Documentación de las funciones miembro	132
5.13.4. Documentación de los datos miembro	132
5.14. Referencia de la Clase OperadorAdaptacionProbMutacion	133
5.14.1. Descripción detallada	133
5.14.2. Documentación del constructor y destructor	135
5.14.3. Documentación de las funciones miembro	136
5.14.4. Documentación de los datos miembro	137
5.15. Referencia de la Clase OperadorCruce	139
5.15.1. Descripción detallada	139
5.15.2. Documentación del constructor y destructor	140
5.15.3. Documentación de las funciones miembro	140
5.16. Referencia de la Clase OperadorCruceArreglo	142
5.16.1. Descripción detallada	142
5.16.2. Documentación del constructor y destructor	143

5.16.3. Documentación de las funciones miembro	143
5.16.4. Documentación de los datos miembro	144
5.17. Referencia de la Clase OperadorCruceBoolDiscreto	145
5.17.1. Descripción detallada	145
5.17.2. Documentación del constructor y destructor	146
5.17.3. Documentación de las funciones miembro	146
5.18. Referencia de la Clase OperadorCruceEnteroAritmetico	147
5.18.1. Descripción detallada	147
5.18.2. Documentación del constructor y destructor	148
5.18.3. Documentación de las funciones miembro	149
5.18.4. Documentación de los datos miembro	149
5.19. Referencia de la Clase OperadorCruceEnteroBLX	151
5.19.1. Descripción detallada	151
5.19.2. Documentación del constructor y destructor	152
5.19.3. Documentación de las funciones miembro	153
5.19.4. Documentación de los datos miembro	153
5.20. Referencia de la Clase OperadorCruceEnteroDiscreto	155
5.20.1. Descripción detallada	155
5.20.2. Documentación del constructor y destructor	156
5.20.3. Documentación de las funciones miembro	156
5.21. Referencia de la Clase OperadorCruceEnteroHeuristico	157
5.21.1. Descripción detallada	157
5.21.2. Documentación del constructor y destructor	158
5.21.3. Documentación de las funciones miembro	158
5.22. Referencia de la Clase OperadorCruceEnteroIntermedioExtendido	160
5.22.1. Descripción detallada	160
5.22.2. Documentación del constructor y destructor	161
5.22.3. Documentación de las funciones miembro	161
5.23. Referencia de la Clase OperadorCruceEnteroLineal	163

5.23.1. Descripción detallada	163
5.23.2. Documentación del constructor y destructor	164
5.23.3. Documentación de las funciones miembro	164
5.24. Referencia de la Clase <code>OperadorCruceEnteroLinealBGA</code>	166
5.24.1. Descripción detallada	166
5.24.2. Documentación del constructor y destructor	167
5.24.3. Documentación de las funciones miembro	168
5.24.4. Documentación de los datos miembro	168
5.25. Referencia de la Clase <code>OperadorCruceEnteroPlano</code>	169
5.25.1. Descripción detallada	169
5.25.2. Documentación del constructor y destructor	170
5.25.3. Documentación de las funciones miembro	170
5.26. Referencia de la Clase <code>OperadorCruceRealAritmetico</code>	171
5.26.1. Descripción detallada	171
5.26.2. Documentación del constructor y destructor	172
5.26.3. Documentación de las funciones miembro	172
5.26.4. Documentación de los datos miembro	173
5.27. Referencia de la Clase <code>OperadorCruceRealBLX</code>	175
5.27.1. Descripción detallada	175
5.27.2. Documentación del constructor y destructor	176
5.27.3. Documentación de las funciones miembro	177
5.27.4. Documentación de los datos miembro	177
5.28. Referencia de la Clase <code>OperadorCruceRealDiscreto</code>	179
5.28.1. Descripción detallada	179
5.28.2. Documentación del constructor y destructor	180
5.28.3. Documentación de las funciones miembro	180
5.29. Referencia de la Clase <code>OperadorCruceRealHeuristico</code>	181
5.29.1. Descripción detallada	181
5.29.2. Documentación del constructor y destructor	182

5.29.3. Documentación de las funciones miembro	182
5.30. Referencia de la Clase <code>OperadorCruceRealIntermedioExtendido</code> .	184
5.30.1. Descripción detallada	184
5.30.2. Documentación del constructor y destructor	185
5.30.3. Documentación de las funciones miembro	185
5.31. Referencia de la Clase <code>OperadorCruceRealLineal</code>	187
5.31.1. Descripción detallada	187
5.31.2. Documentación del constructor y destructor	188
5.31.3. Documentación de las funciones miembro	188
5.32. Referencia de la Clase <code>OperadorCruceRealLinealBGA</code>	190
5.32.1. Descripción detallada	190
5.32.2. Documentación del constructor y destructor	191
5.32.3. Documentación de las funciones miembro	192
5.32.4. Documentación de los datos miembro	192
5.33. Referencia de la Clase <code>OperadorCruceRealPlano</code>	193
5.33.1. Descripción detallada	193
5.33.2. Documentación del constructor y destructor	194
5.33.3. Documentación de las funciones miembro	194
5.34. Referencia de la Clase <code>OperadorFinalizacion</code>	195
5.34.1. Descripción detallada	195
5.34.2. Documentación del constructor y destructor	195
5.34.3. Documentación de las funciones miembro	196
5.35. Referencia de la Clase <code>OperadorFinalizacionOffline</code>	197
5.35.1. Descripción detallada	197
5.35.2. Documentación del constructor y destructor	198
5.35.3. Documentación de las funciones miembro	199
5.35.4. Documentación de los datos miembro	200
5.36. Referencia de la Clase <code>OperadorFinalizacionOnline</code>	201
5.36.1. Descripción detallada	201

5.36.2. Documentación del constructor y destructor	202
5.36.3. Documentación de las funciones miembro	203
5.36.4. Documentación de los datos miembro	204
5.37. Referencia de la Clase OperadorMutacion	205
5.37.1. Descripción detallada	205
5.37.2. Documentación del constructor y destructor	207
5.37.3. Documentación de las funciones miembro	207
5.37.4. Documentación de los datos miembro	208
5.38. Referencia de la Clase OperadorMutacionArreglo	209
5.38.1. Descripción detallada	209
5.38.2. Documentación del constructor y destructor	211
5.38.3. Documentación de las funciones miembro	211
5.38.4. Documentación de los datos miembro	212
5.39. Referencia de la Clase OperadorMutacionBoolUniforme	213
5.39.1. Descripción detallada	213
5.39.2. Documentación del constructor y destructor	214
5.39.3. Documentación de las funciones miembro	215
5.39.4. Documentación de los datos miembro	216
5.40. Referencia de la Clase OperadorMutacionEnteroMuhlenbein	217
5.40.1. Descripción detallada	217
5.40.2. Documentación del constructor y destructor	219
5.40.3. Documentación de las funciones miembro	219
5.40.4. Documentación de los datos miembro	221
5.41. Referencia de la Clase OperadorMutacionEnteroNoUniforme	222
5.41.1. Descripción detallada	222
5.41.2. Documentación del constructor y destructor	224
5.41.3. Documentación de las funciones miembro	225
5.41.4. Documentación de los datos miembro	226
5.42. Referencia de la Clase OperadorMutacionEnteroUniforme	228

5.42.1. Descripción detallada	228
5.42.2. Documentación del constructor y destructor	229
5.42.3. Documentación de las funciones miembro	230
5.42.4. Documentación de los datos miembro	231
5.43. Referencia de la Clase OperadorMutacionRealMuhlenbein	232
5.43.1. Descripción detallada	232
5.43.2. Documentación del constructor y destructor	234
5.43.3. Documentación de las funciones miembro	234
5.43.4. Documentación de los datos miembro	236
5.44. Referencia de la Clase OperadorMutacionRealNoUniforme	237
5.44.1. Descripción detallada	237
5.44.2. Documentación del constructor y destructor	239
5.44.3. Documentación de las funciones miembro	240
5.44.4. Documentación de los datos miembro	241
5.45. Referencia de la Clase OperadorMutacionRealUniforme	243
5.45.1. Descripción detallada	243
5.45.2. Documentación del constructor y destructor	244
5.45.3. Documentación de las funciones miembro	245
5.45.4. Documentación de los datos miembro	246
5.46. Referencia de la Clase OperadorParejas	247
5.46.1. Descripción detallada	247
5.46.2. Documentación del constructor y destructor	248
5.46.3. Documentación de las funciones miembro	248
5.47. Referencia de la Clase OperadorParejasAdyacentes	249
5.47.1. Descripción detallada	249
5.47.2. Documentación del constructor y destructor	250
5.47.3. Documentación de las funciones miembro	250
5.48. Referencia de la Clase OperadorParejasAleatorias	251
5.48.1. Descripción detallada	251

5.48.2. Documentación del constructor y destructor	252
5.48.3. Documentación de las funciones miembro	252
5.49. Referencia de la Clase OperadorParejasExtremos	253
5.49.1. Descripción detallada	253
5.49.2. Documentación del constructor y destructor	254
5.49.3. Documentación de las funciones miembro	254
5.50. Referencia de la Clase OperadorProbabilidad	255
5.50.1. Descripción detallada	255
5.50.2. Documentación del constructor y destructor	256
5.50.3. Documentación de las funciones miembro	256
5.51. Referencia de la Clase OperadorProbabilidadHomogenea	257
5.51.1. Descripción detallada	257
5.51.2. Documentación del constructor y destructor	258
5.51.3. Documentación de las funciones miembro	258
5.52. Referencia de la Clase OperadorProbabilidadLineal	259
5.52.1. Descripción detallada	259
5.52.2. Documentación del constructor y destructor	260
5.52.3. Documentación de las funciones miembro	261
5.52.4. Documentación de los datos miembro	261
5.53. Referencia de la Clase OperadorProbabilidadProporcional	263
5.53.1. Descripción detallada	263
5.53.2. Documentación del constructor y destructor	264
5.53.3. Documentación de las funciones miembro	264
5.54. Referencia de la Clase OperadorReproduccion	266
5.54.1. Descripción detallada	266
5.54.2. Documentación del constructor y destructor	267
5.54.3. Documentación de las funciones miembro	267
5.55. Referencia de la Clase OperadorReproduccionCruceSimple	268
5.55.1. Descripción detallada	268

5.55.2. Documentación del constructor y destructor	269
5.55.3. Documentación de las funciones miembro	269
5.56. Referencia de la Clase OperadorReproduccionDosPadresDosHijos	270
5.56.1. Descripción detallada	270
5.56.2. Documentación del constructor y destructor	271
5.56.3. Documentación de las funciones miembro	271
5.57. Referencia de la Clase OperadorReproduccionMejoresEntre- PadresEHijos	272
5.57.1. Descripción detallada	272
5.57.2. Documentación del constructor y destructor	273
5.57.3. Documentación de las funciones miembro	273
5.58. Referencia de la Clase OperadorReproduccionMejorPadreMejorHijo	274
5.58.1. Descripción detallada	274
5.58.2. Documentación del constructor y destructor	275
5.58.3. Documentación de las funciones miembro	275
5.59. Referencia de la Clase OperadorSeleccion	276
5.59.1. Descripción detallada	276
5.59.2. Documentación del constructor y destructor	277
5.59.3. Documentación de las funciones miembro	277
5.60. Referencia de la Clase OperadorSeleccionEstocasticaRemplazo .	278
5.60.1. Descripción detallada	278
5.60.2. Documentación del constructor y destructor	279
5.60.3. Documentación de las funciones miembro	279
5.61. Referencia de la Clase Poblacion	280
5.61.1. Descripción detallada	280
5.61.2. Documentación del constructor y destructor	282
5.61.3. Documentación de las funciones miembro	282
5.61.4. Documentación de los datos miembro	285

6.1. Referencia del Archivo arreglos.h	287
6.2. Referencia del Archivo genarreglo.h	288
6.2.1. Documentación de las definiciones	291
6.2.2. Documentación de los tipos definidos	293
6.3. Referencia del Archivo genbool.h	295
6.3.1. Documentación de las definiciones	295
6.4. Referencia del Archivo genentero.h	297
6.4.1. Documentación de las definiciones	299
6.5. Referencia del Archivo genetico.h	300
6.5.1. Documentación de las definiciones	306
6.5.2. Documentación de las enumeraciones	309
6.5.3. Documentación de las funciones	309
6.6. Referencia del Archivo genreal.h	311
6.6.1. Documentación de las definiciones	313
6.7. Referencia del Archivo UNGenetico.h	314
6.8. Referencia del Archivo ventana.h	315
6.8.1. Documentación de las definiciones	316
6.8.2. Documentación de las enumeraciones	317
A. Valores por defecto de UNGenético	318
A.1. Valores por defecto de la clase AlgoritmoGenetico	318
A.2. Operadores por defecto de UNGenético	320
A.3. Operadores de UNGenético	321
B. GNU Library General Public License, Version 2	323
B.0.1. GNU LIBRARY GENERAL PUBLIC LICENSE	326

Capítulo 1

UNGenético

1.1. Descripción

UNGenético es una librería desarrollada en lenguaje C++ que permite optimizar modelos a partir de la implementación de algoritmos genéticos. Usando esta librería es posible plantear y modelar una amplia variedad de sistemas gracias a que ha sido diseñada para codificar genomas híbridamente, es decir, *UNGenético* permite almacenar distintos tipos de genes como booleanos, enteros y reales en un mismo genoma así como arreglos de genes de tamaño variable de estos mismos tipos. Adicionalmente, *UNGenético* posee distintas opciones para ejecutar algoritmos genéticos gracias a que cuenta con múltiples operadores genéticos adaptables al tipo de gen y/o al modelo implementado.

UNGenético tiene como propósito principal despertar el interés de la comunidad académica internacional en la implementación de los algoritmos genéticos como estrategia computacional alternativa y eficiente para la optimización de sistemas y procesos complejos en los que el uso de herramientas tradicionales no es adecuado.

UNGenético 2.0 se ha diseñado con el fin de facilitar su utilización, esta versión contiene modificaciones importantes con respecto a la primera que proporcionan nuevas opciones de menor complejidad para la creación de proyectos de optimización. Complementariamente, *UNGenético 2.0* está acompañada por nuevas herramientas gráficas para la ayuda en la construcción y ejecución de proyectos de optimización como *UNGenético Wizard* y *UNGenético Graphics* que mantienen el propósito primordial de compatibilidad en distintas plataformas presente desde la primera versión de la librería.

1.2. Antecedentes

UNGenético es un proyecto desarrollado en el Departamento de Ingeniería Eléctrica de la Universidad Nacional de Colombia como parte de un proceso investigativo en el área de la computación flexible.

La primera versión fue publicada en el año 2002 por el ingeniero Oscar Duarte Velasco Ph.D., ésta ya incorporaba las características de codificación híbrida y compatibilidad para diferentes sistemas operativos.

La segunda versión de *UNGenético* se realizó en el año 2004 como proyecto de grado de Andrés Delgadillo Vega, Sebastián Madrid Arroyo y Jorge Mario Vélez Gutiérrez para optar al título de ingenieros electricistas.

En esta segunda versión se optimizó la estructura de clases de la librería, se aumentó considerablemente el número de operadores genéticos y se implementaron un conjunto de instrucciones que facilitan la creación de modelos de optimización. Complementariamente al trabajo realizado en la librería, para *UNGenético 2.0* se crearon aplicaciones gráficas como *UNGenético Wizard* y *UNGenético Graphics* además de una documentación y un manual de usuario que colaboran en la creación y ejecución de proyectos de optimización.

1.3. Licencia

UNGenético 2.0 es una librería en C++ de algoritmos genéticos con codificación híbrida. Copyright (C) 2004 Andrés Delgadillo Vega, Sebastián Madrid Arroyo y Jorge Mario Vélez Gutiérrez.

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

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

GNU Library General Public License, Version 2 (Anexo B).

1.4. Créditos

UNGenético V2.0 fue desarrollado por Andrés Delgadillo Vega, Sebastián Madrid Arroyo y Jorge Mario Vélez Gutiérrez miembros del departamento de Ingeniería Eléctrica de la Universidad Nacional de Colombia bajo la dirección del ingeniero Oscar Duarte Velasco Ph.D.

Capítulo 2

Utilización de UNGenético 2.0

En este capítulo se tratarán los puntos básicos para la construcción de un proyecto de optimización usando UNGENÉTICO 2.0. Es importante aclarar que estos proyectos pueden ser contruidos para ser visualizados en dos formas distintas: la primera y más sencilla corresponde a la aplicación estándar de los compiladores de lenguaje C++ conocida como *consola*, la segunda corresponde al entorno gráfico ofrecido por la librería *WXWINDOWS*, donde se pueden crear aplicaciones en diferentes plataformas siempre y cuando se implementen un conjunto de instrucciones propias de esta librería. En las siguientes secciones se explicará el método de construcción para los dos tipos de aplicaciones, cuando así se requiera.

Con el fin de ilustrar el proceso plenamente, en la sección 2.1 se ha planteado una función prototipo para la cual se buscará el valor del óptimo global. Mientras que en las demás secciones se explica cada etapa de la construcción del proyecto utilizando UNGENÉTICO 2.0.

2.1. Sistema propuesto

Como ejemplo se propone la maximización de la función

$$f(x, y, z) = \sum_{i=1}^k x + 4y + \sin(15z)e^{-z}$$

Donde,

$$x \in \{0, 1\}, \text{ con } k = 1, \dots, 10.$$

$y \in \mathbb{Z}[-10, 40]$ tal que y es impar

$z \in \mathbb{R}[0, 6]$

La optimización de esta función sugiere la utilización de codificación híbrida ya que cada una de sus variables es de distinto tipo; además la función cuenta con una restricción para una de ellas, por lo tanto se justifica el uso de UNGENÉTICO 2.0 como herramienta de optimización para esta función.

2.2. Creación del proyecto

Las optimizaciones implementadas en UNGENÉTICO 2.0 se ejecutan dentro de un proyecto propio de un compilador de lenguaje C++. Para incluir UNGENÉTICO 2.0, en la carpeta del proyecto deben copiarse todos los archivos que hacen parte de la librería, tanto los encabezados como los fuentes (extensiones *.h* y *.cpp*); en caso contrario, se debe especificar en el proyecto la ruta donde se encuentran dichos archivos. El proyecto solamente debe contener por lo menos un archivo fuente de C++ (extensión *.cpp*) donde se especificarán todas las características del sistema a optimizar, su nombre es de libre escogencia.

2.2.1. Creación del proyecto en consola

En la mayoría de compiladores de C++ es posible crear proyectos especiales para aplicaciones en consola. Por lo tanto, dependiendo del compilador utilizado se debe crear un proyecto que construya este tipo de aplicación.

En el archivo principal del proyecto, es necesario emplear una instrucción que incluya el archivo *UNGenetico.h*, encargado de vincular a los demás archivos de la librería. El siguiente fragmento muestra cómo debe iniciarse el archivo mencionado para el sistema propuesto.

```
#include "UNGenetico.h"
```

2.2.2. Creación del proyecto usando entorno gráfico

Para crear una aplicación gráfica es necesario crear un proyecto que incluya la librería wxWINDOWS, estos proyectos cambian su forma de creación y configuración en cada compilador donde se implemente, por lo tanto es recomendable consultar la documentación de esta librería para crear este tipo de proyectos.

En el archivo principal del proyecto, debe realizarse la definición de la constante *USAR_VENTANA*, con el fin de utilizar la interfaz gráfica diseñada para UNGENÉTICO 2.0. Adicionalmente, es necesario incluir el archivo *UNGenetico.h*. El siguiente fragmento muestra cómo debe iniciar el archivo principal para el sistema propuesto usando entorno gráfico.

```
#define USAR_VENTANA
#include "UNGenetico.h"
```

2.3. Definición de las propiedades del algoritmo genético

Para todo proyecto de optimización es necesario definir una clase derivada de *AlgoritmoGenetico* en la que se establecen las propiedades particulares del sistema a optimizar tales como los operadores genéticos, las características de los individuos de la población, la función objetivo, entre otras.

Las macros *DECLARAR_ALGORITMO(NombreAlgoritmo)* y *FIN_DECLARAR_ALGORITMO* son útiles para definir una clase derivada de *AlgoritmoGenetico* cuyo nombre se puede asignar libremente. Al utilizarlas, implícitamente se está declarando la clase y su constructor por defecto. Dentro de esta definición es posible, si se requiere, sobrecargar los métodos que cambian las propiedades por defecto de la clase, estos son:

- Método *inicializarParametros()*: modifica los valores establecidos por defecto de los miembros de la clase *AlgoritmoGenetico*.
- Método *definirOperadores()*: define los operadores genéticos que se usarán en el proyecto.

Dentro de la definición de esta clase se deben declarar las variables del sistema que coincidirán con los genes que conformarán a los individuos de la población; opcionalmente para este ejemplo se declara también el método *mostrarIndividuo(Individuo& Ind)* que servirá para visualizar la información genética presente en un objeto de la clase *Individuo*.

En el siguiente fragmento se muestra el uso de la macros mencionadas para definir una clase derivada de *AlgoritmoGenetico* llamada *MiAlgoritmoGenetico*, también se declaran las variables *x*, *y*, y *z* del tipo adecuado para cumplir con las condiciones establecidas en el sistema propuesto.

```

DECLARAR_ALGORITMO(MiAlgoritmoGenetico)
    void inicializarParametros();
    void definirOperadores();
    ArregloBool x;
    long y;
    double z;
    void mostrarIndividuo(Individuo& Ind);
FIN_DECLARAR_ALGORITMO

```

2.3.1. Modificación de las propiedades de la clase Algoritmo-Genetico

La clase *AlgoritmoGenetico* posee por defecto un conjunto de propiedades típicas para procesos de optimización. Estas propiedades pueden ser observadas en el anexo [A.1](#). Al implementar el método *inicializarParametros()* declarado anteriormente es posible redefinir los parámetros de la clase *AlgoritmoGenetico* de los cuales se desee cambiar su valor por defecto.

Para el sistema actual se desea:

- Maximizar la función objetivo.
- Ampliar la cantidad de individuos por generación del algoritmo genético a 200.
- Asignar valores iniciales no aleatorios a los genes de la primera generación del algoritmo.

El siguiente fragmento muestra cómo se realizan los cambios mencionados:

```

void MiAlgoritmoGenetico::inicializarParametros()
{
    m_IndicadorMaximizar=true;
    m_TamanoPoblacion=200;
    m_IndicadorInicializarPoblacionAleatoria=false;
}

```

2.3.2. Modificación de los operadores del algoritmo genético

UNGENÉTICO 2.0 posee un conjunto de operadores de probabilidad, selección, asignación de parejas, reproducción, cruce, mutación, adaptación y finalización

para ejecutar el algoritmo genético. El anexo [A.2](#) señala cuáles son los operadores establecidos por defecto. Para definir otro conjunto de operadores debe implementarse el método *definirOperadores()* de la clase *AlgoritmoGenetico*. En la implementación de este método es recomendable utilizar las macros que definen los operadores que utilizará el algoritmo. Los dos tipos de macros diseñadas para este fin son:

- Macros con el prefijo *DEFINIR_*: hacen referencia a operadores de los que sólo debe definirse uno para operar en el algoritmo genético, es decir, los operadores de probabilidad, selección, asignación de parejas y reproducción. Estas son:

```
DEFINIR_OPERADOR_PROBABILIDAD(tipoOperador)
DEFINIR_OPERADOR_SELECCION(tipoOperador)
DEFINIR_OPERADOR_PAREJAS(tipoOperador)
DEFINIR_OPERADOR_REPRODUCCION(tipoOperador)
```

- Macros con el prefijo *ADICIONAR_*: hacen referencia a operadores que pueden actuar en conjunto en el algoritmo, como los operadores de adaptación y finalización, o los correspondientes a cada gen del modelo como los operadores de cruce y mutación. Estas son:

```
ADICIONAR_OPERADOR_ADAPTACION(tipoOperador)
ADICIONAR_OPERADOR_FINALIZACION(tipoOperador)
ADICIONAR_OPERADOR_MUTACION(tipoOperador)
ADICIONAR_OPERADOR_CRUCE(tipoOperador)
```

En todas ellas, el parámetro *tipoOperador* especifica el tipo de operador a utilizar en cada caso junto con sus parámetros iniciales.

El siguiente fragmento muestra cómo definir un nuevo conjunto de operadores para el algoritmo, mientras que en el anexo [A.3](#) se puede observar una lista general de los operadores que hacen parte de la librería y que pueden reemplazar a los operadores asignados por defecto.

```
void MiAlgoritmoGenetico::definirOperadores()
{
    DEFINIR_OPERADOR_PROBABILIDAD(OperadorProbabilidadProporcional)
    DEFINIR_OPERADOR_SELECCION( OperadorSeleccionEstocasticaRemplazo)
    DEFINIR_OPERADOR_PAREJAS(OperadorParejasAdyacentes)
    DEFINIR_OPERADOR_REPRODUCCION(OperadorReproduccionMejorPadreMejorHi j
```

```

ADICIONAR_OPERADOR_ADAPTACION(OperadorAdaptacionProbMutacion
    (this,ADAPTACION_PROBMUTACION_OFFLINE))
ADICIONAR_OPERADOR_FINALIZACION(OperadorTerminacionOffline)

ADICIONAR_OPERADOR_MUTACION(OperadorMutacionArregloBool)
ADICIONAR_OPERADOR_CRUCE(OperadorCruceArregloBool)
ADICIONAR_OPERADOR_MUTACION(OperadorMutacionEnteroNoUniforme(this))
ADICIONAR_OPERADOR_CRUCE(OperadorCruceEnteroHeuristico)
ADICIONAR_OPERADOR_MUTACION(OperadorMutacionRealMuhlenbein)
ADICIONAR_OPERADOR_CRUCE(OperadorCruceRealBLX)
}

```

En algunas de las anteriores definiciones se especifica la palabra clave *this* como parámetro del operador. Esto es necesario en operadores que requieren una referencia al objeto *AlgoritmoGenetico* sobre el que operan.

Cuando se requiera cambiar algún operador de cruce y/o mutación por defecto para algún gen, es indispensable adicionar operadores de cruce y/o mutación para todos los genes que conforman al individuo teniendo en cuenta que el orden en que se adicionen estos operadores debe coincidir con el orden y tipo de los genes del individuo.

2.3.3. Definición de las propiedades de los individuos

Los individuos del algoritmo deben ser definidos de acuerdo con las variables del sistema a optimizar. Con este fin se implementa el método *codificacion(Individuo * Ind, int estado)*. Adicionalmente, para este ejemplo en particular se ha definido el método *mostrarIndividuo(Individuo & Ind)* que se utilizará para visualizar la información genética de los individuos durante la ejecución del algoritmo.

2.3.3.1. Método *codificacion(Individuo * Ind, int estado)*. La codificación consiste en transferir el dato presente en cada variable del modelo hacia su gen respectivo dentro del individuo. La implementación de este método utiliza las macros especializadas en insertar genes del tipo adecuado dentro del individuo que simultáneamente se relacionarán con una variable del sistema.

Las macros mencionadas llevan el prefijo *ADICIONAR_GEN*, y continúan con el tipo de gen que insertan en el individuo, todas ellas reciben los siguientes parámetros, en su orden:

- Individuo: referencia al individuo en el que se adiciona el gen.

- Posición: posición del gen dentro del individuo.
- Variable: variable del sistema asociada con el gen.
- Valor mínimo: valor mínimo que puede tomar el gen. No aplica para arreglos de tipo *bool* ni para arreglos de genes de tipo *bool*.
- Valor máximo: valor máximo que puede tomar el gen. No aplica para arreglos de tipo *bool* ni para arreglos de genes de tipo *bool*.
- Longitud mínima: longitud mínima del arreglo de genes. Sólo aplica para genes de tipo arreglo.
- Longitud máxima: longitud máxima del arreglo de genes. Sólo aplica para genes de tipo arreglo.
- Valor inicial: valor que toma el gen en la primera generación. Siempre debe especificarse, pero sólo se usa cuando no se generan valores iniciales aleatorios en los genes (*m_IndicadorInicializarPoblaciónAleatoria = false*).

Este procedimiento se observa en el siguiente fragmento:

```
void MiAlgoritmoGenetico::codificacion(Individuo * Ind , int estado)
{
    ADICIONAR_GENARREGLO_BOOL(Ind, 0, x, 1, 10 )
    ADICIONAR_GENENTERO(Ind, 1, y, -10, 40, 20)
    ADICIONAR_GENREAL(Ind, 2, z, 0.00, 6.00, 1.0)
}
```

2.3.3.2. Método mostrarIndividuo(Individuo & Ind) en consola. Este método se emplea para visualizar el valor de cada gen presente en el individuo apuntado por *Ind* junto con su función objetivo. El siguiente fragmento muestra el procedimiento realizado para el sistema planteado:

```
void MiAlgoritmoGenetico::mostrarIndividuo(Individuo & Ind)
{
    cout << "\nFuncion Objetivo:\t" << Ind.objetivo(true);
    cout << "\nx: ";
    int tam = x.getSize();
    for(int i=0; i<tam; i++)
    {
        cout << x[i];
    }
}
```

```

        cout << "\ny: " << y;
        cout << "\nz: " << z;
    }

```

En la primera línea de esta función, la instrucción *Ind.objetivo(true)* recibe el parámetro *true* para indicar que deben actualizarse las variables del sistema, puesto que se van a imprimir. Si no se utilizara esta instrucción, o no se quisiera acceder a la función objetivo, sería necesario extraer los valores presentes dentro de cada gen con ayuda del método *codificacion(&Ind, ESTADO_DECODIFICAR)* quien ejecuta la decodificación, es decir, transfiere la información presente en los genes del individuo a las variables del sistema.

2.3.3.3. Método mostrarIndividuo(Individuo * Ind) usando entorno gráfico.

Para la utilización de la interfaz gráfica, los valores de las variables y la función objetivo deben ser almacenados en una cadena de caracteres de tipo *wxstring* propia de la librería *WXWINDOWS*, que en este caso se ha llamado *Cad*. Gracias a la función *AppendText(Cad)*, esta cadena se agregará a un objeto de control de texto apuntado por *m_pTextCtrl* que pertenece a la ventana donde se muestra el proceso del algoritmo. Este procedimiento puede observarse en el siguiente fragmento:

```

void MiAlgoritmoGenetico::mostrarIndividuo(Individuo & Ind)
{
    wxString Cad;
    Cad.Empty();
    Cad << "\nFuncion Objetivo:\t" << Ind.objetivo(true);
    Cad << "\nx: ";
    int tam = x.GetSize();
    for(int i=0; i<tam; i++)
    {
        Cad << x[i];
    }
    Cad << "\ny: " << y;
    Cad << "\nz: " << z;
    m_pFrame->m_pTextCtrl->AppendText(Cad);
}

```

2.3.4. Definición de la función objetivo

En el método *objetivo()* de la clase *AlgoritmoGenetico* se define la función a optimizar, este método retorna un valor real (de tipo *double*) que debe estar relacio-

nado por medio de la función objetivo con las variables del sistema previamente declaradas. Cuando existan restricciones para esta función, debe penalizarse a los individuos que las incumplan, la penalización debe reflejarse en su función objetivo.

Para el sistema planteado en este caso, el método *objetivo()* retorna la variable *FO* correspondiente al valor de la función objetivo para un individuo. En este caso, se requiere maximizar la función objetivo, por lo tanto, los individuos que incumplan la restricción de valores pares para la variable *y*, serán penalizados disminuyendo su función objetivo en 100. Lo anterior puede resumirse en el siguiente fragmento:

```
double MiAlgoritmoGenetico::objetivo()
{
    double FO=0.0;
    int tam = x.getSize();
    for(int i=0; i<tam; i++)
    {
        FO = FO + x[i];
    }
    FO = FO + 4*y + sin(15*z)*exp(-z);
    if(y%2==0)
        FO = FO - 100;
    return (FO);
}
```

2.4. Definición del procedimiento principal en consola

El procedimiento principal que enmarca al algoritmo genético se establece en la función *main()* propia del lenguaje C++. Dentro de este procedimiento es necesario realizar varias actividades, entre ellas:

- Se debe crear una instancia de la clase derivada de *AlgoritmoGenetico* que ha sido definida. Para este ejemplo se ha creado un objeto de la clase *MiAlgoritmoGenetico* llamado *MiAg*.
- El algoritmo genético puede ejecutarse en un sólo paso o mediante un ciclo que recorra cada una de sus generaciones. La primera opción es la más sencilla y se implementa al invocar el método *optimizar()* de la clase

AlgoritmoGenetico. La optimización paso a paso debe iniciar invocando al método *iniciarOptimizacion()* y continúa con un ciclo que repite la ejecución del método *iterarOptimizacion()* hasta que la función *finalizar()* retorne *true*.

- Cuando el algoritmo es ejecutado paso a paso, es posible visualizar la información genética del mejor individuo presente en cada generación. Para esto se utiliza el método *mostrarIndividuo(Individuo & Ind)* creado para este ejemplo. En este caso el método recibirá como parámetro una referencia al mejor individuo de cada generación del algoritmo.
- Si se requiere, se puede invocar la función *mostrarMedidas()* para obtener los valores de las medidas propias del algoritmo cuando éste termine su ejecución.

El siguiente fragmento muestra la implementación del procedimiento principal en consola para el sistema tratado.

```
void main()
{
    MiAlgoritmoGenetico MiAg;
    int i=0;
    MiAg.iniciarOptimizacion();
    do
    {
        MiAg.iterarOptimizacion();
        cout << "\n\nGENERACION " << i;
        MiAg.mostrarIndividuo(*MiAg.m_pMejorEnEstaGeneracion);
        i++;
    }while(!MiAg.finalizar());
    MiAg.mostrarMedidas();
}
```

2.5. Definición del procedimiento principal usando entorno gráfico

Con el fin de crear una ventana para la aplicación del proyecto de optimización, se debe utilizar la macro *DECLARAR_APLICACION(NombreAplicacion)* la cual define a una clase derivada de la clase *wxApp* propia de la librería *WXWINDOWS*. Con ésta única instrucción, implícitamente se está declarando la clase para la

aplicación, el apuntador a la ventana principal que se utilizará y el método *OnInit()*, el cual debe implementarse posteriormente y es quien recibe el control del programa una vez se ha creado la aplicación.

El método *OnInit()* es equivalente a la función principal de la aplicación, dentro de éste es necesario realizar varias actividades adicionales a las mencionadas en el caso anterior, estas son:

- A través del apuntador *m_pframe* que pertenece a la clase de la aplicación, se debe asignar espacio en memoria para un objeto de la clase *AGFrame* que corresponde a la ventana principal; en la misma instrucción se asignan su título, coordenadas y tamaño.
- Se requiere invocar a los métodos *SetTopWindow()* y *Show()* para que la ventana de aplicación sea visible.
- La macro *EJECUTAR_EVENTOS* ejecuta los eventos pendientes de la aplicación, debe usarse en medio de procesos largos para dar una orden de ejecución a las tareas y evitar bloqueos.
- El método *OnInit()* debe retornar el booleano *true* para evitar inconvenientes en la aplicación.

El siguiente fragmento muestra la implementación de las tareas descritas anteriormente para una aplicación denominada *AGApp*.

```
DECLARAR_APLICACION(AGApp)
bool AGApp::OnInit()
{
    m_pFrame = new AGFrame("UNGenético 2.0", 50, 50, 700, 500);
    SetTopWindow(m_pFrame);
    m_pFrame->Show(true);
    MiAlgoritmoGenetico MiAg(m_pFrame);
    MiAg.iniciarOptimizacion();
    int i=0;
    do
    {
        MiAg.iterarOptimizacion();
        wxString Cad;
        Cad.Empty();
        Cad << "\n\nGENERACION " << i;
        m_pFrame->m_pTextCtrl->AppendText(Cad);
        MiAg.mostrarIndividuo(*MiAg.m_pMejorEnEstaGeneracion);
        i++;
    }
```

```

        EJECUTAR_EVENTOS
    }while(!MiAg.finalizar());
    MiAg.mostrarMedidas();
    return true;
}

```

2.6. Ejecución del proyecto

El paso siguiente al terminar la construcción del archivo principal del proyecto, es su ejecución. Este consiste en compilar el proyecto para verificar su correcta estructura de lenguaje y construir finalmente la aplicación donde se ejecuta el algoritmo genético. Estas actividades son realizadas en distintas formas dependiendo del compilador de lenguaje C++ donde se haya creado el proyecto.

Si se siguieron correctamente las secciones anteriores, entonces en la aplicación escogida se presentará la información genética del mejor individuo para cada generación del algoritmo y al finalizar se mostrarán las medidas propias del algoritmo.

Si se definió el uso de la interfaz gráfica, se obtendrá una ventana como la que se muestra en la figura 2.1. La información que se obtiene en esta ventana, puede ser modificada y salvada como un archivo de texto (extensión *.txt*) escogiendo la opción *Guardar* del menú *Archivo*.

2.7. Resumen gráfico del desempeño del algoritmo

Por defecto UNGENÉTICO 2.0 crea un archivo de texto con los resultados del algoritmo genético llamado *salidas.txt*, este archivo contiene los valores de las medidas propias del algoritmo en las generaciones que han sido seleccionadas.

Como complemento de la librería, se ha creado UNGENÉTICO GRAPHICS, una aplicación diseñada especialmente para describir gráficamente el proceso de optimización. Esta herramienta obtiene sus datos de entrada del archivo de texto creado al ejecutar un proyecto en UNGENÉTICO 2.0, de este modo es posible obtener un registro gráfico de la ejecución del algoritmo sin importar si su ejecución se realizó en consola o en entorno gráfico.

Si el proyecto se ejecutó usando entorno gráfico, UNGENÉTICO GRAPHICS estará incluido en la aplicación del proyecto y obtendrá sus datos de entrada automá-

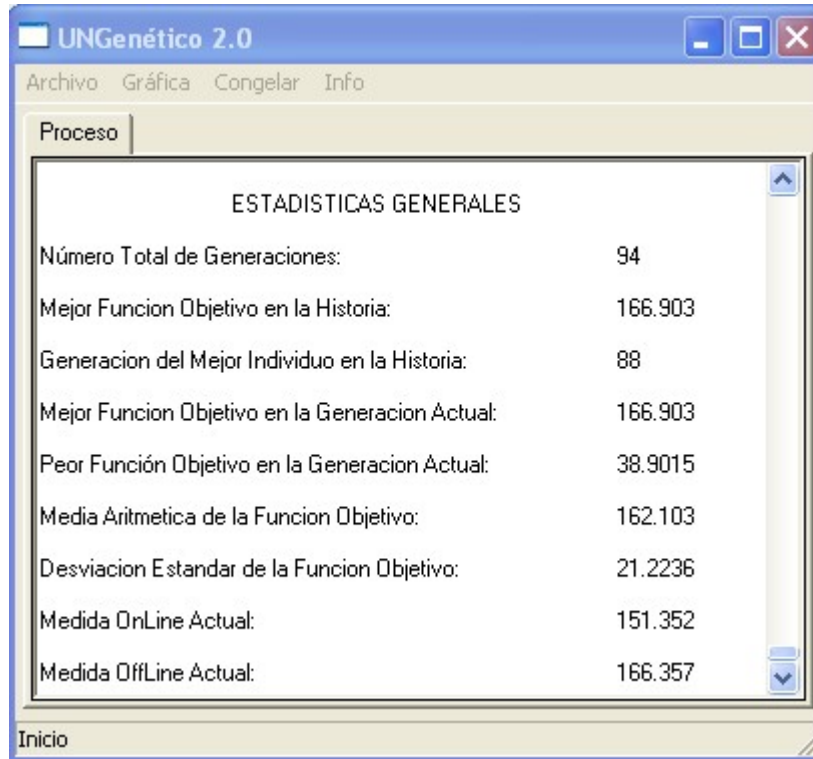


Figura 2.1: Aplicación gráfica de UNGenético 2.0

ticamente del archivo *salidas.txt* que se encuentre en la misma carpeta donde está ubicado el proyecto. Si éste se ejecutó en consola, se deberá abrir de manera independiente la aplicación UNGENÉTICO GRAPHICS e indicar la ruta del archivo de salida del proyecto.

Una vez dentro de la aplicación, al escogerse la opción *Graficar* del menú *Gráfica* se mostrarán las gráficas de las medidas representativas del algoritmo genético en distintas páginas con el siguiente orden:

- *Mejor en la historia*: Gráfica del valor de la función objetivo del mejor individuo que ha aparecido en la historia del algoritmo contra la generación en la que apareció.
- *Generación del mejor en la historia*: Gráfica de la generación en la que apareció el mejor individuo en la historia del algoritmo.
- *Mejor en generación actual*: Gráfica del valor de la función objetivo del mejor individuo en la generación actual.

- *Peor en generación actual*: Gráfica del valor de la función objetivo del peor individuo en la generación actual.
- *Media*: Gráfica del valor de la media aritmética de la función objetivo de los individuos pertenecientes a la generación actual.
- *Desviación estándar*: Gráfica del valor de la desviación estándar de la media aritmética de la función objetivo de los individuos pertenecientes a la generación actual.
- *Medida online*: Gráfica del valor de la medida online del algoritmo contra la generación actual.
- *Medida offline*: Gráfica del valor de la medida offline del algoritmo contra la generación actual.

La gráfica 2.2 ilustra un ejemplo de los resultados gráficos obtenidos para el sistema propuesto. Cada una de estas gráficas puede ser editada cambiando el color del fondo, de los ejes y de la curva al escoger la opción correspondiente en el menú *Gráfica*; complementariamente, estas gráficas pueden ser salvadas como archivos de imagen en formato *png* al seleccionar la opción *Guardar* del menú *Archivo*.

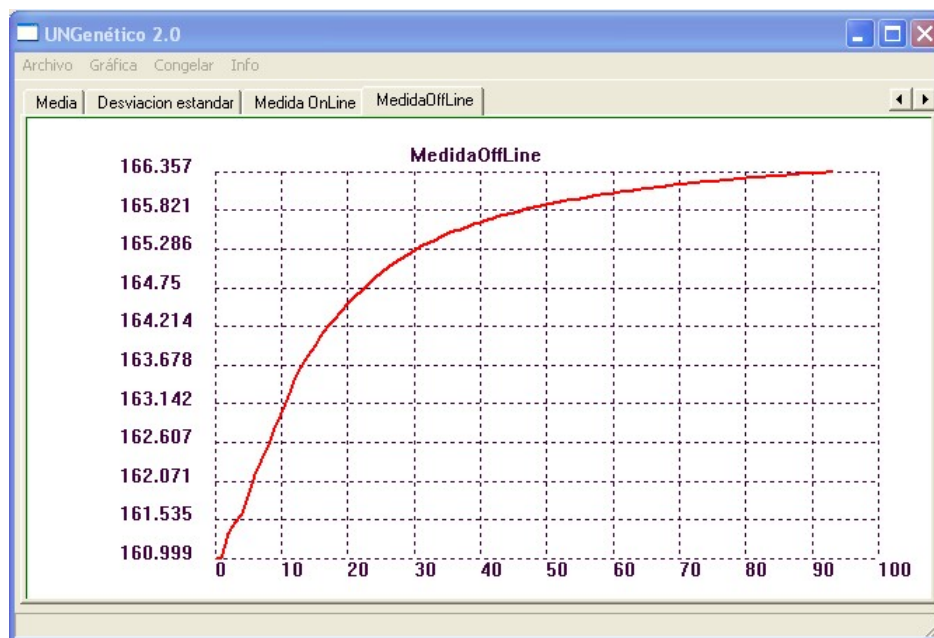


Figura 2.2: Resultados de la optiización obtenidos con UNGenético Graphics

2.8. Utilización de UNGenético Wizard

UNGENÉTICO WIZARD es una aplicación diseñada con el fin de ayudar al usuario en la creación del archivo principal de un proyecto de UNGENÉTICO 2.0 donde se especifican las características más importantes del sistema a optimizar.

Las siguientes secciones describen el procedimiento de creación del archivo principal del proyecto de optimización para el sistema propuesto en la sección [2.1](#) utilizando UNGENÉTICO WIZARD.

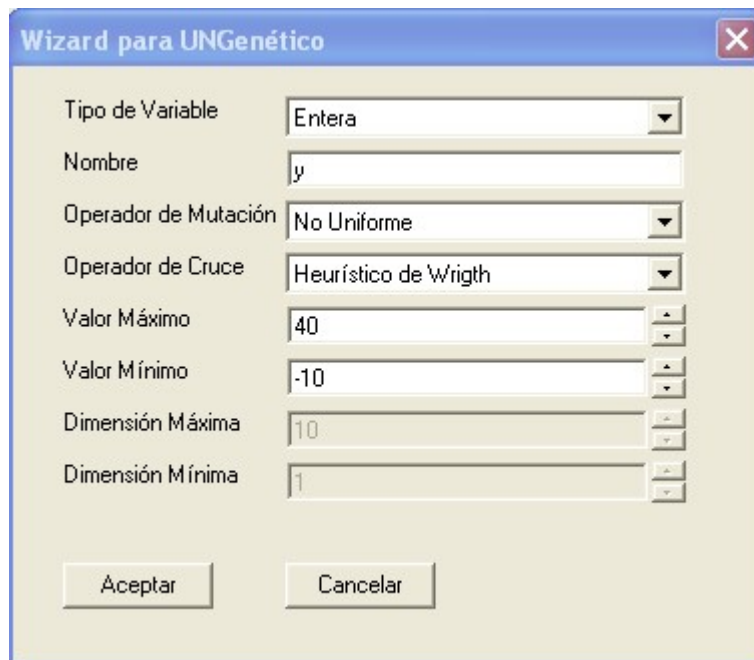
2.8.1. Adición de variables

UNGENÉTICO WIZARD permite adicionar al modelo variables de los tipos que permite manejar la librería. La página *Variables* de la aplicación se especializa en esta labor. Al escoger la opción *Adicionar* se mostrará una ventana que permite especificar las características de las variables del sistema y relacionarlas con un gen, estas son:

- *Tipo de variable*: especifica el tipo de variable del sistema. Puede ser booleana, entera, real o arreglos de estos mismos tipos.
- *Nombre*: identifica a la variable del sistema.
- *Operador de mutación*: define el operador de mutación para el gen relacionado con la variable.
- *Operador de cruce*: define el operador de cruce para el gen relacionado con la variable.
- *Valor máximo*: valor máximo permitido que puede tomar la variable.
- *Valor mínimo*: valor mínimo permitido que puede tomar la variable.
- *Dimensión máxima*: dimensión máxima permitida que puede tomar el arreglo, sólo se puede establecer para variables de tipo arreglo.
- *Dimensión mínima*: dimensión mínima permitida que puede tomar el arreglo, sólo se puede establecer para variables de tipo arreglo.

La figura [2.3](#) muestra las opciones establecidas para la variable y del sistema propuesto, para ella se ha escogido una variable de tipo *Entera* cuyos valores

se han restringido en el rango $[-10, 40]$; también se han escogido los operadores *No Uniforme* y *Heurístico* como operadores de mutación y cruce respectivamente para el gen relacionado con esta variable.



The image shows a software window titled "Wizard para UNGenético". It contains several input fields and dropdown menus for configuring a genetic variable. The fields are as follows:

Field Label	Value
Tipo de Variable	Entera
Nombre	y
Operador de Mutación	No Uniforme
Operador de Cruce	Heurístico de Wrigth
Valor Máximo	40
Valor Mínimo	-10
Dimensión Máxima	10
Dimensión Mínima	1

At the bottom of the window are two buttons: "Aceptar" and "Cancelar".

Figura 2.3: Adición de una variable con UNGenético Wizard

Al finalizar la adición para todas las variables del sistema, éstas quedarán almacenadas junto con todas sus propiedades en la página *Variables* de la aplicación. La figura 2.4 muestra la adición de las variables para el sistema planteado. Puede observarse que la definición de las propiedades es la misma que se realizó manualmente en la sección 2.3.

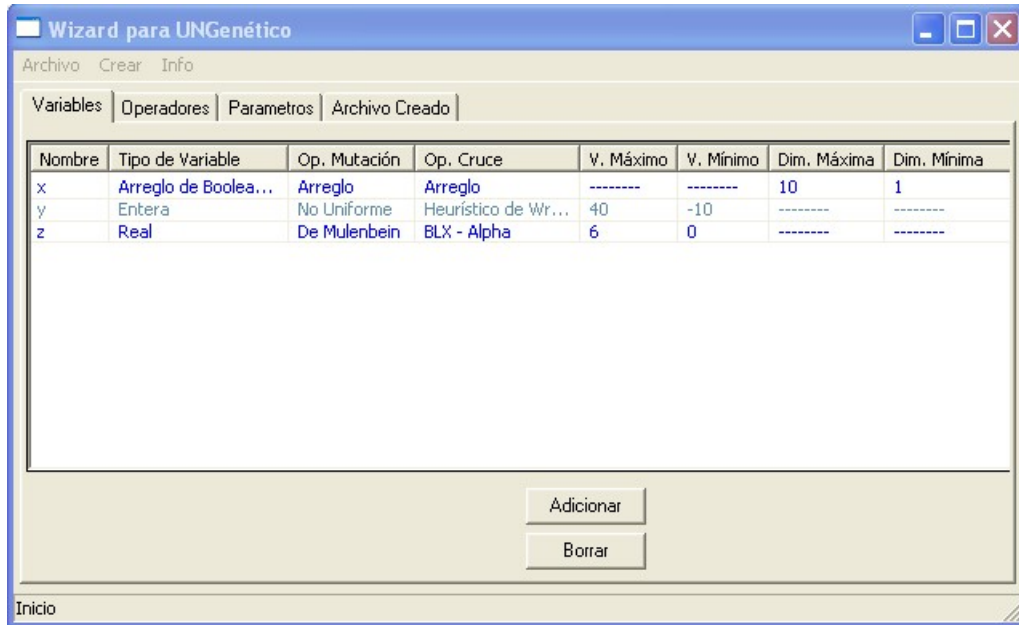


Figura 2.4: Adición de variables en UNGenético Wizard

2.8.2. Definición de operadores genéticos

Con UNGENÉTICO WIZARD también es posible definir los operadores genéticos que se implementarán en el proyecto de optimización. La página *Operadores* ofrece distintas posibilidades para este propósito.

Al activar la opción *Operadores por Defecto*, el proyecto utilizará todos los operadores genéticos establecidos por defecto para la librería (ver Anexo A.2), incluyendo los operadores de cruce y mutación establecidos por defecto para cada tipo de gen; al desactivar esta opción, es posible escoger individualmente el operador genético que se desee utilizar en el proyecto. Para cada caso, UNGENÉTICO WIZARD mostrará todas las posibles opciones.

La figura 2.5 muestra los operadores genéticos escogidos para el sistema tratado.

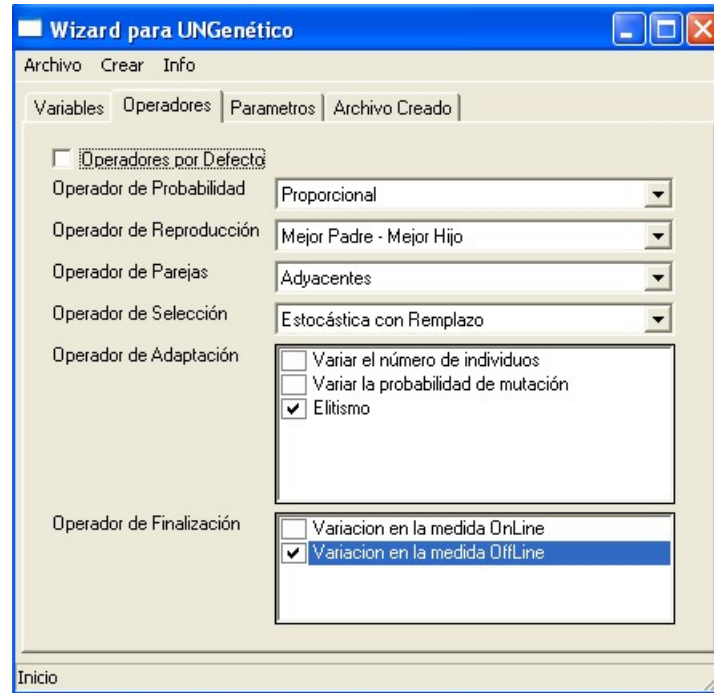


Figura 2.5: Definición de operadores genéticos con UNGenético Wizard

2.8.3. Definición de parámetros para la optimización

UNGENÉTICO WIZARD permite modificar los parámetros del algoritmo genético que definen el número de individuos por generación y el número máximo de iteraciones del algoritmo; también permite establecer si se debe maximizar o minimizar la función objetivo. Esto es posible desde la página *Parámetros* de la aplicación; si en ésta, la opción *Parámetros por Defecto* se encuentra activa, el proyecto utilizará los valores por defecto establecidos en la clase *AlgoritmoGenetico* (ver anexo ??). En esta página también es posible seleccionar algunos parámetros propios de la librería como el nombre del archivo donde se almacenarán las medidas del desempeño del algoritmo y el intervalo de generaciones en que deben ser guardadas estas medidas.

La figura 2.6 muestra la elección de parámetros para el sistema propuesto, en ella se han establecido 200 individuos por generación del algoritmo y un máximo de 200 iteraciones, también se ha escogido la opción *Maximizar* para encontrar el valor máximo de la función de evaluación.

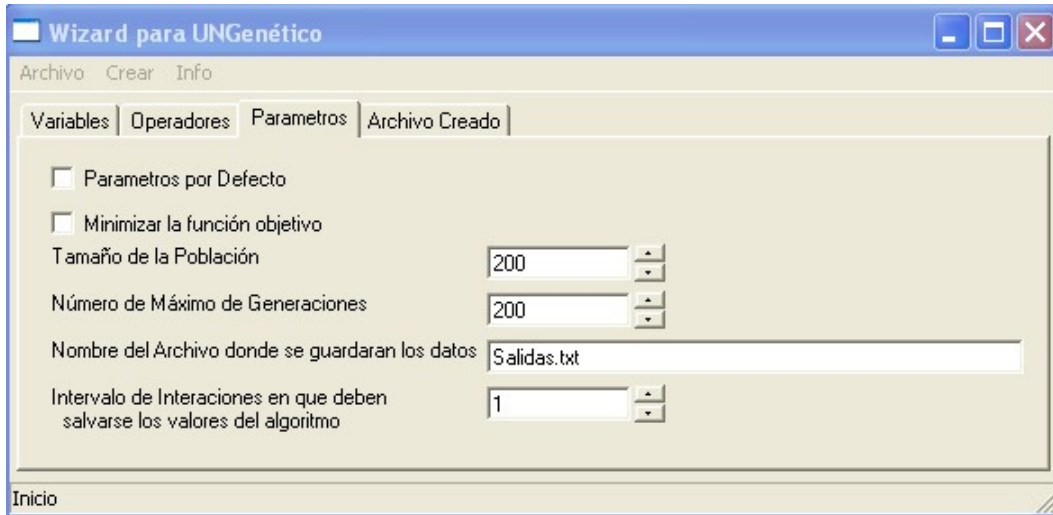


Figura 2.6: Definición de parámetros con UNGenético Wizard

2.8.4. Creación del archivo principal

El siguiente paso luego de definir las propiedades del proyecto de optimización es crear su archivo principal. Esta acción se realiza mediante la opción *Crear* del menú *Crear*; al ejecutarse, la aplicación generará el código en lenguaje C++ del archivo principal del proyecto y lo mostrará en la página *Archivo creado* de la aplicación. Este código puede ser guardado en un archivo fuente de C++ (extensión *.cpp*) seleccionando la opción *Guardar* del menú *Archivo*.

Es importante resaltar que este código solamente contiene información básica acerca del proyecto, el usuario debe complementar y/o modificar este archivo con una función objetivo acorde a su problema a optimizar; si es necesario debe añadir las restricciones pertinentes y las instrucciones propias que se requieran.

Capítulo 3

UNGenético Indice jerárquico

3.1. UNGenético Jerarquía de la clase

Esta lista de herencias esta ordenada rigurosamente, pero no completa, por orden alfabético:

AGFrame	31
AGVentana	41
AlgoritmoGenetico	50
Arreglo	70
Gen	80
GenArreglo	84
GenBool	96
GenEntero	102
GenReal	110
Individuo	118
OperadorAdaptacion	126
OperadorAdaptacionElitismo	128
OperadorAdaptacionNumIndividuos	130
OperadorAdaptacionProbMutacion	133
OperadorCruce	139
OperadorCruceArreglo	142
OperadorCruceBoolDiscreto	145
OperadorCruceEnteroAritmetico	147
OperadorCruceEnteroBLX	151

OperadorCruceEnteroDiscreto	155
OperadorCruceEnteroHeuristico	157
OperadorCruceEnteroIntermedioExtendido	160
OperadorCruceEnteroLineal	163
OperadorCruceEnteroLinealBGA	166
OperadorCruceEnteroPlano	169
OperadorCruceRealAritmetico	171
OperadorCruceRealBLX	175
OperadorCruceRealDiscreto	179
OperadorCruceRealHeuristico	181
OperadorCruceRealIntermedioExtendido	184
OperadorCruceRealLineal	187
OperadorCruceRealLinealBGA	190
OperadorCruceRealPlano	193
OperadorFinalizacion	195
OperadorFinalizacionOffline	197
OperadorFinalizacionOnline	201
OperadorMutacion	205
OperadorMutacionArreglo	209
OperadorMutacionBoolUniforme	213
OperadorMutacionEnteroMuhlenbein	217
OperadorMutacionEnteroNoUniforme	222
OperadorMutacionEnteroUniforme	228
OperadorMutacionRealMuhlenbein	232
OperadorMutacionRealNoUniforme	237
OperadorMutacionRealUniforme	243
OperadorParejas	247
OperadorParejasAdyacentes	249
OperadorParejasAleatorias	251
OperadorParejasExtremos	253
OperadorProbabilidad	255
OperadorProbabilidadHomogenea	257
OperadorProbabilidadLineal	259
OperadorProbabilidadProporcional	263
OperadorReproduccion	266
OperadorReproduccionCruceSimple	268
OperadorReproduccionDosPadresDosHijos	270
OperadorReproduccionMejoresEntrePadresEHijos	272
OperadorReproduccionMejorPadreMejorHijo	274

OperadorSeleccion	276
OperadorSeleccionEstocasticaRemplazo	278
Poblacion	280

Capítulo 4

UNGenético Índice de clases

4.1. UNGenético Lista de componentes

Lista de las clases, estructuras, uniones e interfaces con una breve descripción:

AGFrame (Clase derivada de <i>wxFrame</i> de la librería <i>wxWindows</i> que contiene las ventanas de la clase AGVentana)	31
AGVentana (Clase derivada de <i>wxScrolledWindow</i> de la librería <i>wxWindows</i> que contiene objetos visibles por el usuario)	41
AlgoritmoGenetico (Clase abstracta que administra el proceso de un algoritmo genético)	50
Arreglo (Clase genérica que almacena un arreglo de apuntadores a la clase <i>T</i>)	70
Gen (Clase abstracta que sirve de base a clases que definen genes de distintos tipos)	80
GenArreglo (Clase derivada de la clase Gen , especializada en un gen de tipo arreglo de tamaño variable)	84
GenBool (Clase derivada de la clase Gen especializada en un gen de tipo booleano)	96
GenEntero (Clase derivada de la clase Gen especializada en un gen de tipo entero)	102
GenReal (Clase derivada de la clase Gen especializada en un gen de tipo real)	110
Individuo (Clase que administra la información genética de un individuo)	118
OperadorAdaptacion (Clase abstracta que define el proceso de adaptación del algoritmo genético)	126

OperadorAdaptacionElitismo (Clase derivada de la clase OperadorAdaptacion encargada de efectuar el proceso de elitismo para el algoritmo)	128
OperadorAdaptacionNumIndividuos (Clase derivada de la clase OperadorAdaptacion que hace una variación del número de individuos del algoritmo genético)	130
OperadorAdaptacionProbMutacion (Clase derivada de la clase OperadorAdaptacion que define la estrategia de adaptación para la probabilidad de mutación de los genes de cada individuo de la población)	133
OperadorCruce (Clase abstracta que administra el proceso de cruce entre los individuos del algoritmo genético)	139
OperadorCruceArreglo (Clase derivada de la clase OperadorCruce usada en genes de tipo arreglo. G puede ser GenBool , GenEntero o GenReal . T puede ser bool, long o double)	142
OperadorCruceBoolDiscreto (Clase derivada de la clase OperadorCruce que define el cruce discreto entre dos genes de tipo booleano)	145
OperadorCruceEnteroAritmetico (Clase derivada de la clase OperadorCruce que define un cruce aritmético entre dos genes de tipo entero)	147
OperadorCruceEnteroBLX (Clase derivada de la clase OperadorCruce que define un cruce BLX - α entre dos genes de tipo entero)	151
OperadorCruceEnteroDiscreto (Clase derivada de la clase OperadorCruce que define un cruce discreto entre dos genes de tipo entero)	155
OperadorCruceEnteroHeuristico (Clase derivada de la clase OperadorCruce que define un cruce heurístico entre dos genes de tipo entero)	157
OperadorCruceEnteroIntermedioExtendido (Clase derivada de la clase OperadorCruce que define un cruce extendido intermedio entre dos genes de tipo entero)	160
OperadorCruceEnteroLineal (Clase derivada de la clase OperadorCruce que define un cruce lineal entre dos genes de tipo entero)	163
OperadorCruceEnteroLinealBGA (Clase derivada de la clase OperadorCruce que define un cruce BGA lineal entre dos genes de tipo entero)	166
OperadorCruceEnteroPlano (Clase derivada de la clase OperadorCruce que define un cruce plano entre dos genes de tipo entero)	169
OperadorCruceRealAritmetico (Clase derivada de la clase OperadorCruce que define un cruce aritmético entre dos genes de tipo real)	171

OperadorCruceRealBLX (Clase derivada de la clase OperadorCruce que define un cruce BLX - α entre dos genes de tipo real)	175
OperadorCruceRealDiscreto (Clase derivada de la clase OperadorCruce que define un cruce discreto entre dos genes de tipo real)	179
OperadorCruceRealHeuristico (Clase derivada de la clase OperadorCruce que define un cruce heurístico entre dos genes de tipo real)	181
OperadorCruceRealIntermedioExtendido (Clase derivada de la clase OperadorCruce que define el cruce extendido intermedio entre dos genes de tipo real)	184
OperadorCruceRealLineal (Clase derivada de la clase OperadorCruce que define un cruce lineal entre dos genes de tipo real)	187
OperadorCruceRealLinealBGA (Clase derivada de la clase OperadorCruce que define un cruce BGA lineal entre genes de tipo real)	190
OperadorCruceRealPlano (Clase derivada de la clase OperadorCruce que define un cruce plano entre dos genes de tipo real)	193
OperadorFinalizacion (Clase abstracta que define la estrategia de finalización del algoritmo genético)	195
OperadorFinalizacionOffline (Clase derivada de la clase OperadorFinalizacion que define la finalización del algoritmo basándose en su medida online)	197
OperadorFinalizacionOnline (Clase derivada de la clase OperadorFinalizacion que define la finalización del algoritmo basándose en su medida online)	201
OperadorMutacion (Clase abstracta que administra el proceso de mutación en el algoritmo genético)	205
OperadorMutacionArreglo (Clase derivada de la clase OperadorMutacion empleada en genes de tipo arreglo. G puede ser GenBool , GenEntero o GenReal . T puede ser bool, long o double)	209
OperadorMutacionBoolUniforme (Clase derivada de la clase OperadorMutacion que efectúa una mutación uniforme sobre un gen de tipo booleano)	213
OperadorMutacionEnteroMuhlenbein (Clase derivada de la clase OperadorMutacion que define una mutación Muhlenbein sobre un gen de de tipo entero)	217
OperadorMutacionEnteroNoUniforme (Clase derivada de la clase OperadorMutacion que define una mutación no uniforme sobre un gen de tipo entero)	222

OperadorMutacionEnteroUniforme (Clase derivada de la clase OperadorMutacion que define una mutación uniforme sobre un gen de tipo entero)	228
OperadorMutacionRealMuhlenbein (Clase derivada de la clase OperadorMutacion que define una mutación Muhlenbein sobre un gen de de tipo real)	232
OperadorMutacionRealNoUniforme (Clase derivada de la clase OperadorMutacion que define una mutación no uniforme sobre un gen de tipo real)	237
OperadorMutacionRealUniforme (Clase derivada de la clase OperadorMutacion que define una mutación uniforme sobre un gen de tipo real)	243
OperadorParejas (Clase abstracta que administra la asignación de parejas para cada individuo de la población)	247
OperadorParejasAdyacentes (Clase derivada de la clase OperadorParejas que define la asignación de parejas adyacentes para los individuos de la población)	249
OperadorParejasAleatorias (Clase derivada de OperadorParejas que define la asignación de parejas aleatorias para los individuos de la población)	251
OperadorParejasExtremos (Clase derivada de la clase OperadorParejas que define la asignación de parejas extremas para los individuos de la población)	253
OperadorProbabilidad (Clase abstracta que administra la asignación de la probabilidad de supervivencia a los individuos de la población)	255
OperadorProbabilidadHomogenea (Clase derivada de la clase OperadorProbabilidad que define el proceso de asignación de probabilidad de supervivencia homogénea)	257
OperadorProbabilidadLineal (Clase derivada de la clase OperadorProbabilidad que define el proceso de asignación de probabilidad de supervivencia lineal)	259
OperadorProbabilidadProporcional (Clase derivada de la clase OperadorProbabilidad que define el proceso de asignación de probabilidad de supervivencia proporcional a cada individuo de la población)	263
OperadorReproduccion (Clase abstracta que define la estrategia general de reproducción del algoritmo)	266
OperadorReproduccionCruceSimple (Clase derivada de la clase OperadorReproduccion que define el cruce simple entre dos individuos de la población)	268

OperadorReproduccionDosPadresDosHijos (Clase derivada de la clase OperadorReproduccion que define la estrategia de reproducción dos padres dos hijos)	270
OperadorReproduccionMejoresEntrePadresEHijos (Clase derivada de la clase OperadorReproduccion que define la estrategia de reproducción mejores entre padres e hijos)	272
OperadorReproduccionMejorPadreMejorHijo (Clase derivada de la clase OperadorReproduccion que define la estrategia de reproducción mejor padre mejor hijo)	274
OperadorSeleccion (Clase abstracta que administra el proceso de selección de individuos en la población)	276
OperadorSeleccionEstocasticaRemplazo (Clase derivada de la clase OperadorSeleccion que define el proceso de selección estocástica con reemplazo de los individuos)	278
Poblacion (Clase que administra la información de los individuos de un algoritmo genético)	280

Capítulo 5

UNGenético Documentación de clases

5.1. Referencia de la Clase AGFrame

```
#include <ventana.h>
```

5.1.1. Descripción detallada

Clase derivada de *wxFrame* de la librería *wxWindows* que contiene las ventanas de la clase [AGVentana](#).

Un objeto de esta clase contiene una barra de menú, donde el usuario puede seleccionar varias funciones como guardar los archivos creados, crear gráficas, cambiar los colores de las gráficas, detener y restablecer la visualización de la ejecución del algoritmo e información de la aplicación. También contiene ventanas adicionales en las que aparecen el proceso del algoritmo y las gráficas con las medidas de desempeño del mismo.

Métodos públicos

- [AGFrame](#) (const wxChar *title, int xpos, int ypos, int width, int height)

Constructor de la clase AGFrame.

- `~AGFrame ()`

Destructor de la clase AGFrame.

- `wxWindow * ObtenerPagina (int ID)`

Método que retorna la página identificada por ID.

- `void AgregarPagina (AGVentana *p_pagina, wxString nombre)`

Método que inserta una nueva página en el control m_notebook.

- `bool SetPagina (int ID)`

Método que fija la página identificada por ID.

Atributos públicos

- `wxTextCtrl * m_pTextCtrl`

Apuntador al control de texto donde se muestran los datos seleccionados para cada iteración del algoritmo.

- `wxNotebook * m_pNotebook`

Apuntador al objeto wxNoteBook que contiene las páginas que muestran las gráficas creadas.

- `AGVentana * m_pPagina`

Apuntador al objeto derivado de AGVentana que contiene las gráficas creadas de la ejecución del algoritmo genético.

- `wxString m_NomArch`

Cadena de caracteres correspondiente al nombre del archivo de donde se extraen los valores que serán graficados.

- bool `m_grafica`

Variable que indica si las gráficas ya han sido creadas.

Tipos privados

- enum `menus` {
 `MENU_FILE_SAVE` = 1000, `MENU_FILE_QUIT`,
 `MENU_INFO_ABOUT`, `MENU_GRAFICA_FONDO`,
 `MENU_GRAFICA_GRILLA`, `MENU_GRAFICA_LINEA`,
 `MENU_GRAFICA_GRAFCAR`, `MENU_CONGELAR`,
 `NOTEBOOK` }

Cada valor identifica a un menu de AGFrame.

Métodos privados

- void `OnMenuFondo` (wxCommandEvent &event)
Evento que cambia el color de fondo de la gráfica.
- void `OnMenuGrilla` (wxCommandEvent &event)
Evento que cambia el color de los ejes de la gráfica.
- void `OnMenuLinea` (wxCommandEvent &event)
Evento que cambia el color de la línea de la gráfica.
- void `OnMenuGrafica` (wxCommandEvent &event)
Evento que crea las gráficas en cada página.
- void `OnMenuFileSave` (wxCommandEvent &event)

Evento que guarda una gráfica.

- void **OnMenuFileQuit** (wxCommandEvent &event)

Evento que salva una gráfica.

- void **OnMenuInfoAbout** (wxCommandEvent &event)

Evento que muestra un cuadro de información de la aplicación.

- void **OnClose** (wxCloseEvent &event)

Evento que destruye la aplicación.

- void **OnCongelar** (wxCloseEvent &event)

Evento que detiene la presentación de la información en el control de texto sin detener la ejecución del algoritmo.

- void **SelColor** (wxColour &color)

Método que abre el cuadro de dialogo de selección de colores.

Atributos privados

- wxNotebookSizer * **m_pSizerNB**

Objeto que ajusta el tamaño del objeto notebook.

- wxBoxSizer * **m_pSizerFrame**

Objeto que ajusta el tamaño del frame.

- wxPanel * **m_pPanel**

Apuntador a la ventana que contiene al objeto notebook.

- bool **m_congelar**

Variable que indica si se encuentra detenida o no la actualización de información en el control de texto.

- wxMenuBar * [m_pMenuBar](#)

Apuntador a la barra de menus que se encuentra en el borde superior de la ventana.

- wxMenu * [m_pFileMenu](#)

Apuntador al menu Archivo.

- wxMenu * [m_pInfoMenu](#)

Apuntador al menu Información.

- wxMenu * [m_pMenuGrafica](#)

Apuntador al menu Gráfica.

- wxMenu * [m_pCongelarMenu](#)

Apuntador al menu Congelar.

5.1.2. Documentación de las enumeraciones miembro de la clase

5.1.2.1. enum [menus](#) [private] Cada valor identifica a un menu de AG-Frame.

Valores de la enumeración:

MENU_FILE_SAVE

MENU_FILE_QUIT

MENU_INFO_ABOUT

MENU_GRAFICA_FONDO

MENU_GRAFICA_GRILLA

MENU_GRAFICA_LINEA
MENU_GRAFICA_GRAFCAR
MENU_CONGELAR
NOTEBOOK

5.1.3. Documentación del constructor y destructor

5.1.3.1. **AGFrame** (`const wxChar * title`, `int xpos`, `int ypos`, `int width`, `int height`) Constructor de la clase AGFrame.

En el constructor se crean todos los objetos gráficos tales como barras de menú, barra de estado y ventanas que hacen parte de la aplicación

Parámetros:

title nombre que se mostrará en la barra de título.

xpos posición horizontal de la ventana.

ypos posición vertical de la ventana.

width ancho de la ventana.

height alto de la ventana.

5.1.3.2. **~AGFrame** () [`inline`] Destructor de la clase AGFrame.

5.1.4. Documentación de las funciones miembro

5.1.4.1. **void AgregarPagina** (**AGVentana** * *p_pagina*, `wxString nombre`)

Método que inserta una nueva página en el control *m_notebook*.

Parámetros:

p_pagina apuntador que especifica la nueva página a insertar.

nombre cadena que especifica el título de la nueva página.

5.1.4.2. wxWindow * ObtenerPagina (int *ID*) Método que retorna la página identificada por *ID*.

Realiza una búsqueda en todas las páginas contenidas en *m_notebook*, y retorna la página cuyo identificador coincida con *ID*.

Parámetros:

ID Valor entero que identifica a cada página.

5.1.4.3. void OnClose (wxCloseEvent & event) [private] Evento que destruye la aplicación.

Invoca el método *Destroy()* para destruir todos los objetos descendientes del frame luego de que han sido procesados con el fin de evitar problemas en la aplicación.

5.1.4.4. void OnCongelar (wxCloseEvent & event) [private] Evento que detiene la presentación de la información en el control de texto sin detener la ejecución del algoritmo.

5.1.4.5. void OnMenuFileQuit (wxCommandEvent & event) [private] Evento que salva una gráfica.

5.1.4.6. void OnMenuFileSave (wxCommandEvent & event) [private] Evento que guarda una gráfica.

Salva la gráfica o el texto que se encuentra en la ventana actual

5.1.4.7. void OnMenuFondo (wxCommandEvent & event) [private] Evento que cambia el color de fondo de la gráfica.

Abre un dialogo para cambiar el color guardado en *ColBrush* y vuelve a realizar la gráfica

5.1.4.8. void OnMenuGrafica (wxCommandEvent & event) [private] Evento que crea las gráficas en cada página.

Se encarga de revisar cuántas columnas y datos existen en el archivo correspondiente a *NomArch* y grafica estos datos en páginas diferentes al invocar el método *GrafTexto()*

5.1.4.9. void OnMenuGrilla (wxCommandEvent & event) [private]

Evento que cambia el color de los ejes de la gráfica.

Abre un dialogo para cambiar el color definido en *ColGrilla* y vuelve a realizar la gráfica

5.1.4.10. void OnMenuInfoAbout (wxCommandEvent & event) [private]

Evento que muestra un cuadro de información de la aplicación.

Se ejecuta al escoger la opción *Acerca de* del menú de ayuda, muestra información acerca de los autores de la aplicación.

5.1.4.11. void OnMenuLinea (wxCommandEvent & event) [private]

Evento que cambia el color de la línea de la gráfica.

Abre un dialogo para cambiar el color definido en *ColGraf* y vuelve a realizar la gráfica

5.1.4.12. void SelColor (wxColour & color) [private] Método que abre el cuadro de dialogo de selección de colores.

Parámetros:

color Objeto de define el color a modificar.

5.1.4.13. bool SetPagina (int ID) Método que fija la página identificada por *ID*.

Fija el foco en la página identificada con *ID* retornando *true* si se encontro la página deseada

Parámetros:

ID valor entero que identifica a cada página.

5.1.5. Documentación de los datos miembro

5.1.5.1. bool m_congelar [private] Variable que indica si se encuentra detenida o no la actualización de información en el control de texto.

5.1.5.2. bool [m_grafica](#) Variable que indica si las gráficas ya han sido creadas.

5.1.5.3. wxString [m_NomArch](#) Cadena de caracteres correspondiente al nombre del archivo de donde se extraen los valores que serán graficados.

5.1.5.4. wxMenu* [m_pCongelarMenu](#) [private] Apuntador al menu Congelar.

5.1.5.5. wxMenu* [m_pFileMenu](#) [private] Apuntador al menu Archivo.

5.1.5.6. wxMenu* [m_pInfoMenu](#) [private] Apuntador al menu Información.

5.1.5.7. wxMenuBar* [m_pMenuBar](#) [private] Apuntador a la barra de menus que se encuentra en el borde superior de la ventana.

5.1.5.8. wxMenu* [m_pMenuGrafica](#) [private] Apuntador al menu Gráfica.

5.1.5.9. wxNotebook* [m_pNotebook](#) Apuntador al objeto *wxNoteBook* que contiene las páginas que muestran las gráficas creadas.

5.1.5.10. [AGVentana](#)* [m_pPagina](#) Apuntador al objeto derivado de [AGVentana](#) que contiene las gráficas creadas de la ejecución del algoritmo genético.

5.1.5.11. wxPanel* [m_pPanel](#) [private] Apuntador a la ventana que contiene al objeto notebook.

5.1.5.12. wxBoxSizer* [m_pSizerFrame](#) [private] Objeto que ajusta el tamaño del frame.

5.1.5.13. wxNotebookSizer* m_pSizerNB [private] Objeto que ajusta el tamaño del objeto notebook.

5.1.5.14. wxTextCtrl* m_pTextCtrl Apuntador al control de texto donde se muestran los datos seleccionados para cada iteración del algoritmo.

La documentación para esta clase fué generada a partir de los siguientes archivos:

- [ventana.h](#)
- [ventana.cpp](#)

5.2. Referencia de la Clase AGVentana

```
#include <ventana.h>
```

5.2.1. Descripción detallada

Clase derivada de *wxScrolledWindow* de la librería *wxWindows* que contiene objetos visibles por el usuario.

Al establecer la utilización del ambiente gráfico se crean dos tipos de ventana, una que contiene un control de texto en el que aparece la ejecución del algoritmo, y otras que contienen las gráficas de desempeño del algoritmo genético. Sin embargo, el usuario puede crear una clase derivada de AGVentana y adicionarle cualquier control que pertenezca a la librería *wxWindows*.

Métodos públicos

- **AGVentana** (*wxWindow* *parent, *wxWindowID* id=-1, const *wxPoint* &pos=*wxDefaultPosition*, const *wxSize* &size=*wxDefaultSize*, long style=0)

Constructor de la clase AGVentana.

- **~AGVentana** ()

Destructor de la clase AGVentana.

- *wxWindow* * **ObtenerFrame** ()

Método que retorna el objeto de la clase wxFrame en el que está contenido.

- void **GrafTexto** ()

Método que realiza gráficas descriptivas de datos almacenados en un archivo de texto.

Atributos públicos

- wxTextCtrl * [m_pTxtConsola](#)

Apuntador al control de texto donde se almacena la información seleccionada para cada iteración del algoritmo.

- wxBitmap * [m_pBitmap](#)

Apuntador al objeto donde se almacenan las gráficas creadas.

- wxColour [m_ColBrush](#)

Objeto que define el color de la línea de las gráficas creadas.

- wxColour [m_ColGrilla](#)

Objeto que define el color de los ejes de las gráficas creadas.

- wxColour [m_ColGraf](#)

Objeto que define el color del fondo de las gráficas creadas.

- wxString [m_NameArch](#)

Cadena de caracteres donde se almacena el nombre del archivo de texto del que se obtienen los datos para crear las gráficas.

- wxString [m_Titulo](#)

Cadena de caracteres donde se almacena el nombre de la columna de datos que serán graficados.

- int [m_Ini](#)

Variable que indica la línea del archivo NameArch donde inician los datos que se van a graficar.

- int [m_Columna](#)

Variable que indica la columna del archivo NameArch correspondiente a los datos que se van a graficar.

Métodos protegidos

- void **OnPaint** (wxPaintEvent &event)

Método que actualiza las ventanas que contienen las gráficas creadas.

- void **OnMaxLen** (wxCommandEvent &event)

Método que elimina la información presente en m_TxtConsola cuando su capacidad de almacenamiento llega al límite.

- void **GrafGrilla** ()

Método que crea los ejes para las gráficas creadas.

- long **Extremo** (long valor)

Método que halla el límite superior del eje horizontal de las gráficas creadas.

- double **ObtenerY** (int n, int col)

Método que valida los valores del archivo de texto para ser graficados.

- void **ObtenerValoresGrafica** ()

Método que obtiene los valores del archivo de texto para ser graficados.

- void **ObtenerExtremos** ()

Método que obtiene y valida los extremos inferior y superior del eje vertical de cada gráfica creada.

Atributos protegidos

- wxMemoryDC [m_dib](#)

Dispositivo propio de wxWindows que provee un medio para realizar una gráfica en un mapa de bits.

- wxClientDC [m_dibCopia](#)

Dispositivo propio de wxWindows que permite mostrar la gráfica creada en una ventana.

- int [m_ancho](#)

Variable que indica el ancho de la gráfica.

- int [m_alto](#)

Variable que indica el alto de la gráfica.

- double * [m_pValores](#)

Apuntador a un arreglo de números reales donde se almacenan los valores que serán graficados.

- wxTextFile * [m_pTextarch](#)

Apuntador al archivo de texto de donde se extraen los valores que serán graficados.

- wxSizer * [m_pSizer](#)

Apuntador al objeto que ajusta el tamaño de los controles contenidos en una ventana.

Tipos privados

- enum [Controles](#) { [TEXT_CONSOLA](#) = 3000, [OTRO](#) }

Enumeración a los controles de la ventana.

Atributos privados

- long `m_tam`

Variable que almacena el número de final de generaciones de la ejecución del algoritmo genético.

- long `m_NumGen`

Variable que almacena el limite superior del eje horizontal de las gráficas creadas.

- double `m_max`

Variable que almacena el máximo valor presente en el arreglo p_valores.

- double `m_min`

Variable que almacena el mínimo valor presente en el arreglo p_valores.

5.2.2. Documentación de las enumeraciones miembro de la clase

5.2.2.1. enum `Controles` [private] Enumeración a los controles de la ventana.

Valores de la enumeración:

TEXT_CONSOLA

OTRO

5.2.3. Documentación del constructor y destructor

5.2.3.1. [AGVentana](#) (wxWindow * *parent*, wxWindowID *id* = -1, const wx-Point & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0) Constructor de la clase AGVentana.

En el constructor se coordinan las propiedades principales de la ventana tales como posición y tamaño.

Parámetros:

parent Apuntador a la ventana principal en el que esta contenida.

id Valor entero que identifica a la ventana.

pos Coordenadas que definen la posición de la ventana.

size Conjunto de valores que definen el tamaño de la ventana.

style Valor entero que define el estilo de la ventana.

5.2.3.2. [~AGVentana](#) () Destructor de la clase AGVentana.

Destruye las gráficas creadas por la aplicación al cerrarla.

5.2.4. Documentación de las funciones miembro

5.2.4.1. long Extremo (long *y*) [protected] Método que halla el límite superior del eje horizontal de las gráficas creadas.

Parámetros:

y número de datos que contiene el archivo de texto en cada columna.

5.2.4.2. void GrafGrilla () [protected] Método que crea los ejes para las gráficas creadas.

Crea los ejes principales y auxiliares tanto verticales como horizontales que se mostrarán junto con las líneas creadas por [GrafTexto\(\)](#)

5.2.4.3. void GrafTexto () Método que realiza gráficas descriptivas de datos almacenados en un archivo de texto.

Los datos son extraídos y validados a través del método [ObtenerValoresGrafica\(\)](#). Para cada columna de datos del archivo se crea un objeto de la clase *wxBitmap* que contiene una línea que indica la variación de los valores de la columna correspondiente.

5.2.4.4. void ObtenerExtremos () [protected] Método que obtiene y valida los extremos inferior y superior del eje vertical de cada gráfica creada.

Cada uno de los datos límite se obtiene del arreglo *p_valores*.

5.2.4.5. wxWindow* ObtenerFrame () [inline] Método que retorna el objeto de la clase *wxFrame* en el que está contenido.

5.2.4.6. void ObtenerValoresGrafica () [protected] Método que obtiene los valores del archivo de texto para ser graficados.

Los datos se extraen del archivo cuyo nombre está almacenado en la cadena *NameArch* y se almacenan en el arreglo llamado *p_valores*

5.2.4.7. double ObtenerY (int *n*, int *col*) [protected] Método que valida los valores del archivo de texto para ser graficados.

Convierte una cadena de caracteres presente en un archivo de texto a un número real.

Parámetros:

n línea del archivo donde se encuentra el dato.

col columna del archivo donde se encuentra el dato.

5.2.4.8. void OnMaxLen (wxCommandEvent & *event*) [protected] Método que elimina la información presente en *m_TxtConsola* cuando su capacidad de almacenamiento llega al límite.

5.2.4.9. void OnPaint (wxPaintEvent & *event*) [protected] Método que actualiza las ventanas que contienen las gráficas creadas.

5.2.5. Documentación de los datos miembro

5.2.5.1. **int `m_alto`** `[protected]` Variable que indica el alto de la gráfica.

5.2.5.2. **int `m_ancho`** `[protected]` Variable que indica el ancho de la gráfica.

5.2.5.3. **wxColour `m_ColBrush`** Objeto que define el color de la línea de las gráficas creadas.

5.2.5.4. **wxColour `m_ColGraf`** Objeto que define el color del fondo de las gráficas creadas.

5.2.5.5. **wxColour `m_ColGrilla`** Objeto que define el color de los ejes de las gráficas creadas.

5.2.5.6. **int `m_Columna`** Variable que indica la columna del archivo *NameArch* correspondiente a los datos que se van a graficar.

5.2.5.7. **wxMemoryDC `m_dib`** `[protected]` Dispositivo propio de *wx-Windows* que provee un medio para realizar una gráfica en un mapa de bits.

5.2.5.8. **wxClientDC `m_dibCopia`** `[protected]` Dispositivo propio de *wx-Windows* que permite mostrar la gráfica creada en una ventana.

5.2.5.9. **int `m_Ini`** Variable que indica la línea del archivo *NameArch* donde inician los datos que se van a graficar.

5.2.5.10. **double `m_max`** `[private]` Variable que almacena el máximo valor presente en el arreglo *p_valores*.

5.2.5.11. **double `m_min`** `[private]` Variable que almacena el mínimo valor presente en el arreglo *p_valores*.

5.2.5.12. wxString m_NameArch Cadena de caracteres donde se almacena el nombre del archivo de texto del que se obtienen los datos para crear las gráficas.

5.2.5.13. long m_NumGen [private] Variable que almacena el limite superior del eje horizontal de las gráficas creadas.

5.2.5.14. wxBitmap* m_pBitmap Apuntador al objeto donde se almacenan las gráficas creadas.

5.2.5.15. wxSizer* m_pSizer [protected] Apuntador al objeto que ajusta el tamaño de los controles contenidos en una ventana.

5.2.5.16. wxTextFile* m_pTextarch [protected] Apuntador al archivo de texto de donde se extraen los valores que serán graficados.

5.2.5.17. wxTextCtrl* m_pTxtConsola Apuntador al control de texto donde se almacena la información seleccionada para cada iteración del algoritmo.

5.2.5.18. double* m_pValores [protected] Apuntador a un arreglo de números reales donde se almacenan los valores que serán graficados.

5.2.5.19. long m_tam [private] Variable que almacena el número de final de generaciones de la ejecución del algoritmo genético.

5.2.5.20. wxString m_Titulo Cadena de caracteres donde se almacena el nombre de la columna de datos que serán graficados.

La documentación para esta clase fué generada a partir de los siguientes archivos:

- [ventana.h](#)
- [ventana.cpp](#)

5.3. Referencia de la Clase AlgoritmoGenetico

```
#include <genetico.h>
```

5.3.1. Descripción detallada

Clase abstracta que administra el proceso de un algoritmo genético.

Un objeto de la clase AlgoritmoGenetico contiene una población, los operadores genéticos y todas las características necesarias para llevar a cabo el algoritmo genético y permitir la interacción entre éste y el usuario. Para utilizarla deben definirse las funciones puramente virtuales *codificación* y *objetivo()*. Para realizar todo el proceso del algoritmo genético en un solo paso debe invocarse la función *optimizar()*. Pueden cambiarse los parámetros y operadores por defecto sobrecargando las funciones *inicializarParametros()* y *definirOperadores()*

Métodos públicos

- *AlgoritmoGenetico* ()

Constructor por defecto.

- virtual *~AlgoritmoGenetico* ()

Destructor.

- virtual void *codificacion* (*Individuo* *pInd,int estado)=0

En esta función deben definirse los tres estados de codificación de un individuo.

- virtual double *objetivo* ()=0

En esta función debe definirse la forma de calcular la función de evaluación.

- virtual void *optimizar* ()

Ejecuta todo el proceso de optimización.

- void **iniciarOptimizacion** ()
Prepara el algoritmo genético para su ejecución.
- void **iterarOptimizacion** ()
Efectúa la siguiente iteración del algoritmo.
- bool **finalizar** ()
Indica si el algoritmo debe finalizar.
- void **finalizarOptimizacion** ()
Finaliza el algoritmo genético.
- void **mostrarMedidas** ()
Presenta en pantalla los resultados intermedios del algoritmo genético.
- void **salvar** ()
Almacena en el archivo de salida los resultados intermedios del algoritmo genético.

Atributos públicos

- **Poblacion** * **m_pPoblacionActual**
Apuntador a la poblacion del algoritmo genético.
- **Individuo** * **m_pMejorEnEstaGeneracion**
Apuntador al mejor individuo de la generación actual.
- **Individuo** * **m_pPeorEnEstaGeneracion**
Apuntador al peor individuo de la generación actual.

- `Individuo * m_pMejorEnLaHistoria`

Apuntador al mejor individuo de la historia del algoritmo.

- `Individuo * m_pModelo`

Apuntador al `Individuo` que se emplea como modelo para producir copias de él.

- `Arreglo< OperadorMutacion > * m_pListaOperadorMutacion`

`Arreglo` de operadores de mutación.

- `Arreglo< OperadorCruce > * m_pListaOperadorCruce`

`Arreglo` de operadores de cruce.

Parámetros del algoritmo

- `long m_GeneracionMaxima`

Número máximo de generaciones.

- `long m_IntervaloSalvar`

Intervalo de iteraciones en que deben salvarse y/o mostrarse los valores intermedios.

- `char m_NombreArchivo [400]`

Nombre del archivo en que se guardan los valores intermedios de las iteraciones.

Indicadores

- `bool m_IndicadorUsarAdaptacion`

Indica si se va a utilizar alguna estrategia de adaptación en el algoritmo genético.

- `bool m_IndicadorInicializarPoblacionAleatoria`

Indica si al inicializar la poblacion se crean individuos aleatorios o con los valores iniciales especificados.

- bool `m_IndicadorMaximizar`

Indica si debe maximizarse o minimizarse la función objetivo.

- bool `m_IndicadorArchivo`

Indica si deben salvarse los resultados intermedios de las iteraciones del algoritmo genético.

- bool `m_IndicadorMostrar`

Indica si deben mostrarse en pantalla los resultados intermedios de las iteraciones del algoritmo genético.

- bool `m_IndicadorMostrarMejorEnHistoria`

Indica si se debe salvar y/o mostrar la función objetivo del mejor individuo en la historia.

- bool `m_IndicadorMostrarGeneracionMejorHistorico`

Indica si se debe salvar y/o mostrar la generación en que apareció el mejor individuo en la historia.

- bool `m_IndicadorMostrarMejorEnGeneracion`

Indica si se debe salvar y/o mostrar la función objetivo del mejor individuo de la generación actual.

- bool `m_IndicadorMostrarPeorEnGeneracion`

Indica si se debe salvar y/o mostrar la función objetivo del peor individuo de la generación actual.

- bool `m_IndicadorMostrarMedia`

Indica si se debe salvar y/o mostrar el promedio de las funciones objetivo de la generación actual.

- bool `m_IndicadorMostrarDesviacion`

Indica si se debe salvar y/o mostrar la desviación estándar de las funciones objetivo de la generación actual.

- bool `m_IndicadorMostrarOnLine`

Indica si se debe salvar y/o mostrar la medida OnLine para la generación actual.

- bool `m_IndicadorMostrarOffLine`

Indica si se debe salvar y/o mostrar la medida OffLine para la generación actual.

Valores propios de cada iteración

- int `m_TamanoPoblacion`

Número de individuos de la generación actual.

- long `m_Generacion`

Número de generación actual.

- long `m_GeneracionDelMejorEnLaHistoria`

Número de generación en la que apareció el mejor individuo de la historia del algoritmo.

- double `m_Media`

Promedio aritmético de las funciones objetivo de los individuos de la generación actual.

- double `m_MedidaOnLine`

Promedio aritmético acumulado de las funciones objetivo de todos los individuos que han existido en la historia del algoritmo.

- double `m_MedidaOffLine`

Promedio aritmético acumulado de las funciones objetivo de los mejores individuos de cada generación en la historia del algoritmo.

- double `m_MedidaOnLineAnterior`

Medida OnLine de la generación anterior.

- double `m_MedidaOffLineAnterior`

Medida OffLine de la generación anterior.

- double `m_Desviacion`

Desviación estándar de las funciones objetivo de los individuos presentes en la generación actual.

Métodos protegidos

- void `inicializarVariables ()`

Inicializa las variables y parámetros del algoritmo.

- void `inicializarApuntadores ()`

Inicializa los apuntadores en posiciones nulas.

- void `crearOperadores ()`

Crea los operadores a utilizar por el algoritmo.

- void `asignarProbabilidad ()`

Efectúa la asignación de probabilidad de supervivencia a los individuos de la población.

- void `seleccionar ()`

Efectúa la selección de los individuos que sobreviven para la nueva generación.

- void **asignarParejas** ()
Efectúa la asignación de parejas de individuos para el cruce.
- void **reproducir** ()
Ejecuta la estrategia general de reproducción.
- void **mutar** ()
Efectúa la mutación de la población actual.
- void **adaptacion** ()
Efectúa la adaptación de parámetros del algoritmo.
- void **actualizarMedidas** ()
Calcula los valores intermedios de las medidas de desempeño del algoritmo genético.

Atributos protegidos

- **OperadorProbabilidad** * **m_pOpProbabilidad**
Apuntador al operador de asignación de probabilidad de supervivencia.
- **OperadorSeleccion** * **m_pOpSeleccion**
Apuntador al operador de selección de individuos.
- **OperadorParejas** * **m_pOpParejas**
Apuntador al operador de asignación de parejas de cruce.
- **OperadorReproduccion** * **m_pOpReproduccion**
Apuntador al operador de reproducción.

- `Arreglo< OperadorAdaptacion > * m_pListaOperadorAdaptacion`

Arreglo de operadores de adaptación del algoritmo.

- `Arreglo< OperadorFinalizacion > * m_pListaOperadorFinalizacion`

Arreglo de operadores de finalización.

Métodos privados

- virtual void `inicializarParametros ()`

Esta función puede sobregargarse para cambiar los valores por defecto del algoritmo.

- virtual void `definirOperadores ()`

Esta función puede sobregargarse para definir operadores diferentes a los establecidos por defecto.

5.3.2. Documentación del constructor y destructor

5.3.2.1. `AlgoritmoGenetico ()` Constructor por defecto.

Inicializa todos los apuntadores de la clase en posiciones nulas

5.3.2.2. `virtual ~AlgoritmoGenetico () [inline, virtual]` Destructor.

Destruye todos los objetos creados. Es virtual para poder definirse en las clases derivadas.

5.3.3. Documentación de las funciones miembro

5.3.3.1. `void actualizarMedidas () [protected]` Calcula los valores intermedios de las medidas de desempeño del algoritmo genético.

Es invocada por *iterarOptimización()*. Actualiza los siguientes miembros:

- *m_Media*
- *m_MedidaOnLine*
- *m_MedidaOffLine*
- *m_Desviacion*
- *m_pMejorEnEstaGeneracion*
- *m_pPeorEnEstaGeneracion*
- *m_pMejorEnLaHistoria*
- *m_GeneracionDelMejorEnLaHistoria*
- *m_MedidaOffLineAnterior*
- *m_MedidaOnLineAnterior*

5.3.3.2. void adaptacion () [protected] Efectúa la adaptación de parámetros del algoritmo.

Sólo se realiza adaptación del algoritmo si *m_IndicadorUsarAdaptacion=true*. Se realiza invocando la función *adaptacion* de los operadores de adaptación del algoritmo en el mismo orden en que se adicionaron en la función *definirOperadores()*. Es invocada por *iterarOptimización()*

5.3.3.3. void asignarParejas () [inline, protected] Efectúa la asignación de parejas de individuos para el cruce.

Invoca la función *asignarParejas* del operador de asignación de parejas del algoritmo. Es invocada por *iterarOptimización()*

5.3.3.4. void asignarProbabilidad () [inline, protected] Efectúa la asignación de probabilidad de supervivencia a los individuos de la población.

Invoca la función *asignarProbabilidad* del operador de probabilidad del algoritmo. Es invocada por *iterarOptimización()*

5.3.3.5. virtual void codificacion (*Individuo* * *pInd*, int *estado*) [pure virtual] En esta función deben definirse los tres estados de codificación de un individuo.

Debe sobrecargarse en las clases derivadas. Esta función es invocada definiendo el estado en que se encuentra el individuo. Cada individuo invoca esta función cuando es necesario realizar su decodificación. También es invocada por *iniciarOptimizacion()* para la creación del individuo modelo. El usuario solamente debe invocarla cuando requiere hacer actualización explícita de las variables del sistema o de la información genética de algún individuo. Existen tres estados: creación, codificación y decodificación, que se determinan mediante el parámetro *estado*, el cual puede tomar los siguientes valores: ESTADO_CREACION, ESTADO_CODIFICACION, y ESTADO_DECODIFICACION.

- ESTADO_CREACION: En el momento de su construcción, se define la forma que adoptará el individuo, determinada por la constitución del genoma, es decir, el número y tipos de genes que contendrá.
- ESTADO_CODIFICACION: Define la forma en que se traducen las variables del modelo en información genética del individuo. Estas variables deben estar definidas en la clase derivada.
- ESTADO_DECODIFICACION: Define la forma en que se traduce la información genética del individuo en las variables del modelo. Estas variables deben estar definidas en la clase derivada.

En la implementación de esta función, pueden utilizarse las MACROS que facilitan la adición de genes al individuo, y que proveen los tres estados en una sola instrucción.

Parámetros:

pInd Apuntador al individuo en proceso de codificación.

estado Estado en que se encuentra el individuo.

5.3.3.6. void crearOperadores () [protected] Crea los operadores a utilizar por el algoritmo.

Realiza las siguientes acciones:

- Crea los arreglos contenedores de los Operadores de mutacion, cruce, adaptación y finalización.

- Invoca la función [definirOperadores\(\)](#) la cual puede ser definida para utilizar operadores diferentes a los definidos por defecto.
- Verifica si fueron creados los operadores correctamente, en caso contrario crea los operadores por defecto.

Los operadores por defecto se pueden ver en [Operadores por defecto de UNGenético 2.0](#) Es invocada por [iniciarOptimizacion\(\)](#)

5.3.3.7. virtual void definirOperadores () [inline, private, virtual] Esta función puede sobregargarse para definir operadores diferentes a los establecidos por defecto.

5.3.3.8. bool finalizar () Indica si el algoritmo debe finalizar.

Invoca los operadores de finalización del algoritmo genético en el mismo orden en que se adicionaron. Si alguno indica que el algoritmo debe finalizar, retorna true. También retorna true siempre que se alcance el número máximo de iteraciones especificado por *m_GeneracionMaxima*. Es invocada por [optimizar\(\)](#).

Devuelve:

true si el algoritmo debe finalizar. *false* en caso contrario.

5.3.3.9. void finalizarOptimizacion () Finaliza el algoritmo genético.

Elimina todos los objetos creados en el algoritmo. Es invocada por [iniciarOptimizacion\(\)](#) y por el destructor de la clase.

5.3.3.10. void inicializarApuntadores () [protected] Inicializa los apuntadores en posiciones nulas.

Es invocada por los constructores de la clase y por [inicializarVariables\(\)](#)

5.3.3.11. virtual void inicializarParametros () [inline, private, virtual] Esta función puede sobregargarse para cambiar los valores por defecto del algoritmo.

Es invocada por [inicializarVariables\(\)](#)

5.3.3.12. void inicializarVariables () [protected] Inicializa las variables y parámetros del algoritmo.

Inicializa los apuntadores en posiciones nulas, inicializa las variables propias de cada iteración y establece los parámetros por defecto. Invoca la función *inicializarParametros()* en la cual usted puede cambiar los parámetros por defecto. Ver parámetros establecidos por defecto en *Valores por defecto de la clase AlgoritmoGenetico*. Es invocada por *iniciarOptimizacion()*.

5.3.3.13. void iniciarOptimizacion () Prepara el algoritmo genético para su ejecución.

Inicializa todas las variables y parámetros, crea el individuo modelo y los operadores a utilizar en el algoritmo. Si el algoritmo está en mitad de proceso, éste es finalizado y preparado para comenzar de nuevo. Es invocada por *optimizar()*.

5.3.3.14. void iterarOptimizacion () Efectúa la siguiente iteración del algoritmo.

Si es la primera iteración genera una nueva población del tamaño definido por *m_TamanoPoblacion*. Una iteración contiene las siguientes instrucciones:

- *asignarProbabilidad();*
- *seleccionar();*
- *asignarParejas();*
- *reproducir();*
- *mutar();*
- *adaptacion();*
- *actualizarMedidas();*

Salva las medidas en archivo y las muestra en pantalla, dependiendo de los valores de *m_IndicadorArchivo*, *m_IndicadorMostrar* y *m_IntervaloSalvar*, invocando las funciones

- *salvar()*
- *mostrarMedidas()*

Es invocada por *optimizar()*

5.3.3.15. void mostrarMedidas () Presenta en pantalla los resultados intermedios del algoritmo genético.

Esta función es invocada por *iterarOptimizacion()* solamente si *m_IndicadorMostrar=true*. Los resultados que se muestran dependen de los indicadores especificados. Se muestran en el siguiente orden.

Valor Mostrado	Depende de
Generación	Siempre es mostrado
Función objetivo del mejor en la historia	m_IndicadorMostrarMejorEnHistoria
Generación del Mejor en la historia	m_IndicadorMostrarGeneracion-MejorHistorico
Mejor Función Objetivo en la Generación Actual	m_IndicadorMostrarMejorEn-Generacion
Peor Función Objetivo en la Generación Actual	m_IndicadorMostrarPeorEn-Generacion
Media Aritmética de la Función Objetivo	m_IndicadorMostrarMedia
Desviación Estándar de la Función Objetivo	m_IndicadorMostrarDesviacion
Medida OnLine Actual	m_IndicadorMostrarOnLine
Medida OffLine Actual	m_IndicadorMostrarOffLine

Cada valor es mostrado si su respectivo indicador tiene valor *true*

5.3.3.16. void mutar () [inline, protected] Efectúa la mutación de la población actual.

Invoca la función *mutar* de la población actual. Es invocada por *iterarOptimización()*

5.3.3.17. virtual double objetivo () [pure virtual] En esta función debe definirse la forma de calcular la función de evaluación.

Debe sobrecargarse en las clases derivadas. La función objetivo se calcula haciendo uso de las variables del modelo, definidas en la clase derivada. El usuario no debe invocar esta función explícitamente. Esta función es invocada internamente por cada individuo para calcular su función objetivo despues de haber invocado la función *codificación(ESTADO_DECODIFICACION)* la cual actualiza las variables del sistema.

Devuelve:

Valor de la función de evaluación para las variables actuales del sistema.

5.3.3.18. void optimizar () [virtual] Ejecuta todo el proceso de optimización.

El proceso completo de optimización se realiza comenzando por la inicialización y ejecutando iteraciones del algoritmo hasta que se llegue al final lo cual es indicado por el criterio de finalización especificado para el algoritmo. El proceso completo de optimización comprende las siguientes instrucciones:

```
iniciarOptimizacion();  
do  
{  
    iterarOptimizacion();  
}while(!finalizar());
```

5.3.3.19. void reproducir () [inline, protected] Ejecuta la estrategia general de reproducción.

Invoca la función *reproducir* del operador de reproducción del algoritmo. Es invocada por *iterarOptimización()*

5.3.3.20. void salvar () Almacena en el archivo de salida los resultados intermedios del algoritmo genético.

Esta función es invocada por *iterarOptimizacion()* solamente si *m_Indicador-Archivo=true*. El archivo de salida es el especificado por *m_NombreArchivo*. Los resultados salvados dependen de los indicadores especificados. Se guardan en el siguiente orden.

Valor Guardado	Depende de
Generación	Siempre es mostrado
Función objetivo del mejor en la historia	m_IndicadorMostrarMejorEnHistoria
Generación del Mejor en la historia	m_IndicadorMostrarGeneracion-MejorHistorico
Mejor Función Objetivo en la Generación Actual	m_IndicadorMostrarMejorEn-Generacion
Peor Función Objetivo en la Generación Actual	m_IndicadorMostrarPeorEn-Generacion
Media Aritmética de la Función Objetivo	m_IndicadorMostrarMedia
Desviación Estándar de la Función Objetivo	m_IndicadorMostrarDesviacion
Medida OnLine Actual	m_IndicadorMostrarOnLine
Medida OffLine Actual	m_IndicadorMostrarOffLine

Cada valor es guardado si su respectivo indicador tiene valor *true*

5.3.3.21. void seleccionar () [*inline, protected*] Efectúa la selección de los individuos que sobreviven para la nueva generación.

Invoca la función *seleccionar* del operador de selección del algoritmo. Es invocada por *iterarOptimización()*

5.3.4. Documentación de los datos miembro

5.3.4.1. double m_Desviacion Desviación estándar de las funciones objetivo de los individuos presentes en la generación actual.

5.3.4.2. long m_Generacion Número de generación actual.

5.3.4.3. long m_GeneracionDelMejorEnLaHistoria Número de generación en la que apareció el mejor individuo de la historia del algoritmo.

5.3.4.4. long m_GeneracionMaxima Número máximo de generaciones.

5.3.4.5. bool m_IndicadorArchivo Indica si deben salvarse los resultados intermedios de las iteraciones del algoritmo genético.

5.3.4.6. bool m_IndicadorInicializarPoblacionAleatoria Indica si al inicializar la poblacion se crean individuos aleatorios o con los valores iniciales especificados.

5.3.4.7. bool m_IndicadorMaximizar Indica si debe maximizarse o minimizarse la función objetivo.

5.3.4.8. bool m_IndicadorMostrar Indica si deben mostrarse en pantalla los resultados intermedios de las iteraciones del algoritmo genético.

5.3.4.9. bool m_IndicadorMostrarDesviacion Indica si se debe salvar y/o mostrar la desviación estándar de las funciones objetivo de la generación actual.

5.3.4.10. bool m_IndicadorMostrarGeneracionMejorHistorico Indica si se debe salvar y/o mostrar la generación en que apareció el mejor individuo en la historia.

5.3.4.11. bool m_IndicadorMostrarMedia Indica si se debe salvar y/o mostrar el promedio de las funciones objetivo de la generación actual.

5.3.4.12. bool m_IndicadorMostrarMejorEnGeneracion Indica si se debe salvar y/o mostrar la función objetivo del mejor individuo de la generación actual.

5.3.4.13. bool m_IndicadorMostrarMejorEnHistoria Indica si se debe salvar y/o mostrar la función objetivo del mejor individuo en la historia.

5.3.4.14. bool m_IndicadorMostrarOffLine Indica si se debe salvar y/o mostrar la medida OffLine para la generación actual.

5.3.4.15. bool m_IndicadorMostrarOnLine Indica si se debe salvar y/o mostrar la medida OnLine para la generación actual.

5.3.4.16. bool m_IndicadorMostrarPeorEnGeneracion Indica si se debe salvar y/o mostrar la función objetivo del peor individuo de la generación actual.

5.3.4.17. bool m_IndicadorUsarAdaptacion Indica si se va a utilizar alguna estrategia de adaptación en el algoritmo genético.

5.3.4.18. long m_IntervaloSalvar Intervalo de iteraciones en que deben salvarse y/o mostrarse los valores intermedios.

5.3.4.19. double m_Media Promedio aritmético de las funciones objetivo de los individuos de la generación actual.

5.3.4.20. double m_MedidaOffLine Promedio aritmético acumulado de las funciones objetivo de los mejores individuos de cada generación en la historia del algoritmo.

5.3.4.21. double m_MedidaOffLineAnterior Medida OffLine de la generación anterior.

5.3.4.22. double m_MedidaOnLine Promedio aritmético acumulado de las funciones objetivo de todos los individuos que han existido en la historia del algoritmo.

5.3.4.23. double m_MedidaOnLineAnterior Medida OnLine de la generación anterior.

5.3.4.24. char m_NombreArchivo[400] Nombre del archivo en que se guardan los valores intermedios de las iteraciones.

5.3.4.25. Arreglo<OperadorAdaptacion>* m_pListaOperadorAdaptacion
[protected] Arreglo de operadores de adaptación del algoritmo.

5.3.4.26. Arreglo<OperadorCruce>* m_pListaOperadorCruce Arreglo de operadores de cruce.

El tipo del operador en cada posición del arreglo debe coincidir con el tipo de gen en la misma posición del genoma del individuo modelo

5.3.4.27. Arreglo<OperadorFinalizacion>* m_pListaOperadorFinalizacion
[protected] Arreglo de operadores de finalización.

5.3.4.28. Arreglo<OperadorMutacion>* m_pListaOperadorMutacion
Arreglo de operadores de mutación.

El tipo del operador en cada posición del arreglo debe coincidir con el tipo de gen en la misma posición del genoma del individuo modelo

5.3.4.29. Individuo* m_pMejorEnEstaGeneracion Apuntador al mejor individuo de la generación actual.

5.3.4.30. Individuo* m_pMejorEnLaHistoria Apuntador al mejor individuo de la historia del algoritmo.

5.3.4.31. Individuo* m_pModelo Apuntador al **Individuo** que se emplea como modelo para producir copias de él.

5.3.4.32. OperadorParejas* m_pOpParejas [protected] Apuntador al operador de asignación de parejas de cruce.

5.3.4.33. OperadorProbabilidad* m_pOpProbabilidad [protected] Apuntador al operador de asignación de probabilidad de supervivencia.

5.3.4.34. OperadorReproduccion* m_pOpReproduccion [protected] Apuntador al operador de reproducción.

5.3.4.35. OperadorSeleccion* m_pOpSeleccion [protected] Apuntador al operador de selección de individuos.

5.3.4.36. Individuo* m_pPeorEnEstaGeneracion Apuntador al peor individuo de la generación actual.

5.3.4.37. Poblacion* m_pPoblacionActual Apuntador a la poblacion del algoritmo genético.

5.3.4.38. int [m_TamanoPoblacion](#) Número de individuos de la generación actual.

La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genetico.h](#)
- [genetico.cpp](#)

5.4. Referencia de la Clase Arreglo

```
#include <arreglos.h>
```

5.4.1. Descripción detallada

template<class T> class Arreglo< T > Clase genérica que almacena un arreglo de apuntadores a la clase T.

El acceso a los objetos del arreglo es al estilo de los arreglos de C y no al estilo de listas encadenadas, lo que le da mayor rapidez. No debe confundirse la capacidad del arreglo con la cantidad de ítems almacenados. En lo posible, para evitar reasignación innecesaria de memoria, antes de adicionar ítems al arreglo debe asignarse la capacidad necesaria para el número de ítems que se prevee se van a adicionar al arreglo. Esto puede hacerse mediante la función *asignarMemoria*.

Métodos públicos

- **Arreglo** (const **Arreglo** &origen)

Constructor por copia de otro arreglo. No se puede usar con arreglos de clases abstractas.

- **Arreglo** ()

Constructor por defecto.

- const **Arreglo**< T > & **operator=** (const **Arreglo**< T > &otro)

Operador de asignación a partir de otro arreglo. No se puede usar con arreglos de clases abstractas.

- virtual **~Arreglo** (void)

Destructor.

- int **Adicionar** (T *pNuevo)

Adiciona un apuntador al final del arreglo y retorna la posición en la que fue adicionado.

- int **Insertar** (T *pNuevo, int pos)

Inserta un apuntador en una posición determinada del arreglo.

- T * **reemplazar** (T *pNuevo, int pos)

Reemplaza el apuntador ubicado en una posición determinada del arreglo por otro apuntador.

- bool **IntercambiarPos** (int pos1, int pos2)

Intercambia los apuntadores de dos posiciones determinadas.

- int **Truncar** (int nuevoTam, bool eliminarObjetos=true, bool liberar_memoria=true)

Reduce el tamaño del arreglo.

- int **getSize** () const

Retorna el número de ítems presentes en el arreglo.

- T * **getPtr** (int pos) const

Retorna el apuntador ubicado en una posición determinada del arreglo.

- T & **getObj** (int pos) const

Retorna una referencia al objeto ubicado en una posición determinada del arreglo.

- T & **operator[]** (int pos) const

Retorna una referencia al objeto ubicado en una posición determinada del arreglo.

- bool **asignarMemoria** (int n)

Aumenta la capacidad del arreglo para albergar un número determinado de ítems.

- void **liberarMemoria** ()

Reduce la capacidad del arreglo al número de ítems actual.

- T * **Detach** (int pos)

Excluye del arreglo el apuntador ubicado en una posición determinada, sin destruir el objeto al que apunta.

- void **Destroy** (int pos)

Excluye del arreglo el apuntador ubicado en una posición determinada, destruyendo el objeto al que apunta.

- void **FlushDetach** ()

Excluye del arreglo todos sus miembros sin destruir los objetos.

- void **FlushDestroy** ()

Excluye del arreglo todos sus miembros y destruye todos los objetos.

Atributos privados

- int **m_capacidad**

- int `m_items`
- T ** `m_pData`

5.4.2. Documentación del constructor y destructor

5.4.2.1. `Arreglo (const Arreglo< T > & origen)` Constructor por copia de otro arreglo. No se puede usar con arreglos de clases abstractas.

Copia idénticamente el arreglo origen, creando copias de los objetos a los que hacen referencia los apuntadores contenidos en él. No se debe usar para arreglos de clases abstractas ya que no se pueden crear instancias de estas clases.

Parámetros:

origen Objeto del que se hace copia.

5.4.2.2. `Arreglo ()` [inline] Constructor por defecto.

Crea un arreglo vacío. No se asigna memoria al arreglo

5.4.2.3. `virtual ~Arreglo (void)` [inline, virtual] Destructor.

Elimina todos los apuntadores y destruye los objetos pertenecientes al arreglo

5.4.3. Documentación de las funciones miembro

5.4.3.1. `int Adicionar (T * pNuevo)` [inline] Adiciona un apuntador al final del arreglo y retorna la posición en la que fue adicionado.

Asigna memoria al arreglo si es necesario.

Parámetros:

pNuevo Apuntador al nuevo miembro del arreglo.

Devuelve:

Posición en la que se adicionó el nuevo miembro.

5.4.3.2. bool asignarMemoria (int n) Aumenta la capacidad del arreglo para albergar un número determinado de ítems.

Es conveniente invocar esta función antes de adicionar nuevos miembros al arreglo cuando es conocido previamente el número de ítems a adicionar ya que evita reasignación de memoria en cada adición de miembros al arreglo.

Parámetros:

n Nueva capacidad total del arreglo. Si el arreglo ya tiene suficiente memoria para albergar el n ítems, nada ocurre.

Devuelve:

true si la memoria fue asignada correctamente. false en caso contrario

5.4.3.3. void Destroy (int pos) Excluye del arreglo el apuntador ubicado en una posición determinada, destruyendo el objeto al que apunta.

Todos los miembros del arreglo que se encuentran en posiciones mayores que pos , disminuyen una posición.

Parámetros:

pos Posición del apuntador a excluir del arreglo.

5.4.3.4. T * Detach (int pos) Excluye del arreglo el apuntador ubicado en una posición determinada, sin destruir el objeto al que apunta.

Todos los miembros del arreglo que se encuentran en posiciones mayores que pos , disminuyen una posición.

Parámetros:

pos Posición del apuntador a excluir del arreglo.

Devuelve:

Apuntador al objeto excluido del arreglo. Si *pos* no es una posición válida retorna NULL

5.4.3.5. void FlushDestroy () [inline] Excluye del arreglo todos sus miembros y destruye todos los objetos.

Libera la memoria ocupada por el arreglo de apuntadores y por cada uno de los objetos apuntados por estos.

5.4.3.6. void FlushDetach () [inline] Excluye del arreglo todos sus miembros sin destruir los objetos.

Libera la memoria ocupada por el arreglo de apuntadores.

5.4.3.7. T& getObj (int *pos*) const [inline] Retorna una referencia al objeto ubicado en una posición determinada del arreglo.

Equivalente al operador [].

Parámetros:

pos Posición del objeto requerido. Si es menor que cero se retorna el objeto en la primera posición. Si es mayor que el número de ítems del arreglo, se retorna el objeto en la última posición.

Devuelve:

Referencia al objeto apuntado por el apuntador ubicado en la posición *pos* del arreglo

5.4.3.8. T* getPtr (int pos) const [inline] Retorna el apuntador ubicado en una posición determinada del arreglo.

Parámetros:

pos Posición del apuntador requerido. Si es menor que cero se retorna el apuntador en la primera posición. Si es mayor que el número de ítems del arreglo, se retorna el apuntador en la última posición.

Devuelve:

Apuntador ubicado en la posición *pos*.

5.4.3.9. int getSize () const [inline] Retorna el número de ítems presentes en el arreglo.

Devuelve:

Tamaño del arreglo

5.4.3.10. int Insertar (T * pNuevo, int pos) Inserta un apuntador en una posición determinada del arreglo.

Todos los miembros a partir de la posición de inserción aumentan una posición. En caso de estar lleno el arreglo, realiza asignación de espacio para 20 ítems más.

Parámetros:

pNuevo Apuntador al nuevo miembro del arreglo.

pos Posición en la que se ubicará el nuevo miembro. Si es menor que cero, es ubicado en la posición cero. Si es mayor que el número de ítems, se adiciona al final del arreglo.

Devuelve:

Posición en la que fué ubicado el nuevo miembro.

5.4.3.11. bool IntercambiarPos (int *pos1*, int *pos2*) [*inline*] Intercambia los apuntadores de dos posiciones determinadas.

El intercambio solo es realizado si ambas posiciones son válidas en el arreglo.

Parámetros:

pos1 Posición en la que será ubicado el apuntador que se encuentra en la posición *pos2*.

pos2 Posición en la que será ubicado el apuntador que se encuentra en la posición *pos1*.

Devuelve:

true si el intercambio es exitoso. false si no se realizó el intercambio.

5.4.3.12. void liberarMemoria () Reduce la capacidad del arreglo al número de ítems actual.

Se utiliza para liberar la memoria no utilizada por el arreglo

5.4.3.13. const Arreglo< T > & operator= (const Arreglo< T > & *origen*)

Operador de asignación a partir de otro arreglo. No se puede usar con arreglos de clases abstractas.

Copia idénticamente el arreglo origen, creando copias de los objetos a los que hacen referencia los apuntadores contenidos en él. No se debe usar para arreglos de clases abstractas ya que no se pueden crear instancias de estas clases.

Parámetros:

origen Objeto del que se hace copia.

Devuelve:

Referencia al arreglo

5.4.3.14. **]** T& operator[] (int *pos*) const [inline]

Retorna una referencia al objeto ubicado en una posición determinada del arreglo.

Es equivalente a getObj(pos). Permite acceder a los miembros del arreglo al estilo del lenguaje C.

Parámetros:

pos Posición del objeto requerido. Si es menor que cero se retorna el objeto en la primera posición. Si es mayor que el número de ítems del arreglo, se retorna el objeto en la última posición.

Devuelve:

Referencia al objeto apuntado por el apuntador ubicado en la posición *pos* del arreglo

5.4.3.15. **T* remplazar (T * *pNuevo*, int *pos*)** [inline] Reemplaza el apuntador ubicado en una posición determinada del arreglo por otro apuntador.

Parámetros:

pNuevo Apuntador que hará parte del arreglo, reemplazando al apuntador ubicado en la posición especificada.

pos Posición del apuntador a reemplazar.

Devuelve:

Apuntador reemplazado. NULL si la posición es inválida

5.4.3.16. **int Truncar (int *nuevoTam*, bool *eliminarObjetos* = true, bool *liberar_memoria* = true)** Reduce el tamaño del arreglo.

Utilice *liberar_memoria*=false si sabe que va a volver a usar la capacidad actual del arreglo para evitar reasignación posterior de memoria.

Parámetros:

nuevoTam Nuevo tamaño del arreglo. Si es mayor que el actual, nada ocurre.

Si es menor que cero, el nuevo tamaño es cero.

eliminarObjetos Si es *true* se eliminan los objetos sobrantes. En caso contrario, los objetos dejan de ser parte del arreglo pero no son eliminados.

liberar_memoria Si es *true* se libera la memoria no necesaria.

Devuelve:

Nuevo tamaño del arreglo.

5.4.4. Documentación de los datos miembro

5.4.4.1. `int m_capacidad` [private]

5.4.4.2. `int m_items` [private]

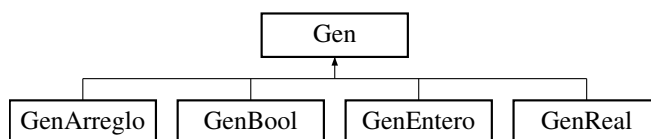
5.4.4.3. `T* * m_pData` [private] La documentación para esta clase fué generada a partir del siguiente archivo:

- `arreglos.h`

5.5. Referencia de la Clase Gen

```
#include <genetico.h>
```

Diagrama de herencias de Gen:



5.5.1. Descripción detallada

Clase abstracta que sirve de base a clases que definen genes de distintos tipos.

Las clases derivadas como [GenBool](#), [GenEntero](#), [GenReal](#), [GenArreglo](#) u otras que cree el usuario son las que establecen la forma en que se almacena y se maneja la información genética. Estas deben implementar todas las funciones puramente virtuales que definen el tratamiento de la información de cada gen. Para cada tipo de gen que se cree deben definirse también operadores de mutación y cruce cuyas clases deben derivarse de las clases [OperadorMutacion](#) y [OperadorCruce](#) respectivamente.

Métodos públicos

- [Gen](#) ()

Constructor por defecto.

- virtual [~Gen](#) ()

Destructor.

- virtual void **generarAleatorio** ()=0

Debe especificar la forma de generar de un gen aleatorio.

- virtual **OperadorMutacion** * **operadorMutacionDefecto** () const=0

Debe especificar el operador de mutación por defecto para el gen.

- virtual **OperadorCruce** * **operadorCruceDefecto** () const=0

Debe especificar el operador de cruce por defecto para el gen.

- virtual **Gen** * **crearCopia** () const=0

Debe generar una copia (creada con el operador new) del objeto Gen actual.

- virtual void **copiar** (const **Gen** &otro)=0

Debe especificar la asignación a partir de otro objeto Gen.

5.5.2. Documentación del constructor y destructor

5.5.2.1. **Gen** () [inline] Constructor por defecto.

Debe sobrecargarse en las clases derivadas si se requiere inicializar algún miembro

5.5.2.2. virtual ~**Gen** () [inline, virtual] Destructor.

Es virtual para poder definirse en las clases derivadas.

5.5.3. Documentación de las funciones miembro

5.5.3.1. virtual void copiar (const [Gen](#) & otro) [pure virtual] Debe especificar la asignación a partir de otro objeto [Gen](#).

Debe sobrecargarse en las clases derivadas. Esta función se podría reemplazar por el operador de asignación (=) si [Gen](#) no fuera una clase abstracta.

Implementado en [GenArreglo](#), [GenBool](#), [GenEntero](#), y [GenReal](#).

5.5.3.2. virtual [Gen](#)* crearCopia () const [pure virtual] Debe generar una copia (creada con el operador *new*) del objeto [Gen](#) actual.

Debe sobrecargarse en las clases derivadas

Devuelve:

Apuntador a un nuevo objeto de la clase derivada, con las mismas características que el actual

Implementado en [GenArreglo](#), [GenBool](#), [GenEntero](#), y [GenReal](#).

5.5.3.3. virtual void generarAleatorio () [pure virtual] Debe especificar la forma de generar de un gen aleatorio.

Debe sobrecargarse en las clases derivadas

Implementado en [GenArreglo](#), [GenBool](#), [GenEntero](#), y [GenReal](#).

5.5.3.4. virtual [OperadorCruce](#)* operadorCruceDefecto () const [pure virtual] Debe especificar el operador de cruce por defecto para el gen.

Debe sobrecargarse en las clases derivadas return Debe retornar un apuntador a un nuevo objeto de una clase derivada de [OperadorCruce](#)

Implementado en [GenArreglo](#), [GenBool](#), [GenEntero](#), y [GenReal](#).

5.5.3.5. virtual [OperadorMutacion](#)* operadorMutacionDefecto () const
[pure virtual] Debe especificar el operador de mutación por defecto para el gen.

Debe sobrecargarse en las clases derivadas return Debe retornar un apuntador a un nuevo objeto de una clase derivada de [OperadorMutacion](#)

Implementado en [GenArreglo](#), [GenBool](#), [GenEntero](#), y [GenReal](#).

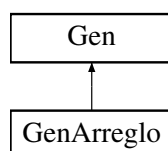
La documentación para esta clase fué generada a partir del siguiente archivo:

- [genetico.h](#)

5.6. Referencia de la Clase GenArreglo

```
#include <genarreglo.h>
```

Diagrama de herencias de GenArreglo:



5.6.1. Descripción detallada

template<class G, class T> class GenArreglo< G, T > Clase derivada de la clase [Gen](#), especializada en un gen de tipo arreglo de tamaño variable.

Se utiliza para representar y almacenar la información genética de variables cuya codificación se realiza con un arreglo de datos que pueden ser enteros, reales o booleanos. G puede ser [GenBool](#), [GenEntero](#) o [GenReal](#). T puede ser bool, long o double.

Métodos públicos

- [GenArreglo](#) (const [GenArreglo](#) &origen)
Constructor por copia.
- [GenArreglo](#) (int TamanoMinimo=1, int TamanoMaximo=10, T ValorMinimo=0, T ValorMaximo=1)
Constructor por defecto.
- [GenArreglo](#) (int TamanoMinimo, int TamanoMaximo, T ValorMinimo, T ValorMaximo, T ValorInicial)

Constructor con ValorInicial.

- `Gen * crearCopia () const`

Crea una copia exacta del gen.

- `void copiar (const Gen &origen)`

Asigna nuevos valores al gen copiando las propiedades de otro gen.

- `const GenArreglo< G, T > & operator= (const GenArreglo< G, T > &origen)`

Operador de asignación a partir de otro objeto GenArreglo<G,T>.

- `const GenArreglo< G, T > & operator= (const Arreglo< T > &origen)`

Operador de asignación a partir de un objeto Arreglo<T>.

- `void convertArreglo (Arreglo< T > &destino) const`

Convierte el objeto GenArreglo<G,T> en un objeto Arreglo<T>.

- `~GenArreglo ()`

Destructor.

- `int setTam (int nuevoTam, bool crearAleatorios=false)`

Cambia el tamaño del arreglo.

- `void generarAleatorio ()`

Genera un arreglo de genes de tamaño y valores aleatorios.

- `OperadorMutacion * operadorMutacionDefecto () const`

Retorna un objeto correspondiente al operador de mutación establecido por defecto para genes de tipo arreglo.

- `OperadorCruce * operadorCruceDefecto () const`

Retorna un objeto correspondiente al operador de cruce establecido por defecto para genes de tipo arreglo.

- `G & getGen (int pos) const`

Retorna el apuntador al gen indicado por pos en el arreglo de genes.

- `T getVal (int pos) const`

Retorna el valor actual del gen en la posición pos del arreglo de genes.

- `bool setVal (int pos, T valor)`

Asigna un nuevo valor al gen ubicado en la posición pos.

- `const GenEntero * getGenItems () const`

Retorna el `GenEntero` que almacena el tamaño del arreglo variable.

- `int getTam () const`

Retorna el tamaño actual del arreglo de genes.

- `int getMinTam () const`

Retorna el tamaño mínimo del arreglo variable.

- `int getMaxTam () const`

Retorna el tamaño máximo del arreglo variable.

- `T getMinVal () const`

Retorna el valor mínimo válido para los datos almacenados en el arreglo de genes.

- T `getMaxVal ()` const

Retorna el valor máximo válido para los datos almacenados en el arreglo de genes.

Métodos protegidos

- G * `crearGen ()` const

Crea un nuevo objeto del tipo de genes contenidos en el arreglo y retorna su apuntador.

Atributos protegidos

- `GenEntero` * `m_pGenItems`

Apuntador al `GenEntero` que almacena el tamaño actual del arreglo.

- `Arreglo` < G > * `m_pArregloGen`

`Arreglo` que contiene la informacion genética, con genes del tipo especificado por G.

- T `m_valMin`

Mínimo valor válido para los datos contenidos en el arreglo.

- T `m_valMax`

Máximo valor válido para los datos contenidos en el arreglo.

- T `m_vallInicial`

Valor inicial para los datos contenidos en el arreglo.

- bool `m_usarVallInicial`

5.6.2. Documentación del constructor y destructor

5.6.2.1. `GenArreglo` (const `GenArreglo`< G, T > & *origen*) Constructor por copia.

Construye el nuevo objeto copiando idénticamente las propiedades de otro objeto de la clase `GenArreglo`.

Parámetros:

origen Objeto del que se hace copia.

5.6.2.2. `GenArreglo` (int *TamanoMinimo* = 1, int *TamanoMaximo* = 10, T *ValorMinimo* = 0, T *ValorMaximo* = 1) Constructor por defecto.

Inicializa todos los miembros de la clase con valores válidos. Asigna a cada gen su valor por defecto. El tamaño inicial del arreglo es el promedio de los límites de tamaño establecidos.

Parámetros:

TamanoMinimo Tamaño mínimo que puede tomar el arreglo, Por defecto es 1.

TamanoMaximo Tamaño máximo que puede tomar el arreglo, Por defecto es 10.

ValorMinimo Valor mínimo que puede tomar cada gen, Por defecto es 0.

ValorMaximo Valor máximo que puede tomar cada gen, Por defecto es 1.

5.6.2.3. `GenArreglo` (`int TamanoMinimo`, `int TamanoMaximo`, `T ValorMinimo`, `T ValorMaximo`, `T ValorInicial`) Constructor con `ValorInicial`.

Inicializa todos los miembros de la clase con valores válidos. Da a cada gen el valor inicial especificado, restringiéndolo a los valores límites. El tamaño inicial del arreglo es el promedio de los límites de tamaño establecidos.

Parámetros:

TamanoMinimo Tamaño mínimo que puede tomar el arreglo,

TamanoMaximo Tamaño máximo que puede tomar el arreglo,

ValorMinimo Valor mínimo que puede tomar cada gen,

ValorMaximo Valor máximo que puede tomar cada gen,

ValorInicial Valor que se asigna a cada gen,

5.6.2.4. `~GenArreglo` () Destructor.

Destruye todos los objetos contenidos en el arreglo

5.6.3. Documentación de las funciones miembro

5.6.3.1. `void convertArreglo (Arreglo< T > & destino) const` Convierte el objeto `GenArreglo<G,T>` en un objeto `Arreglo<T>`.

El arreglo destino toma el mismo tamaño del `GenArreglo`. Los valores del arreglo destino toman los mismos valores de los genes contenidos en el `GenArreglo`, en sus respectivas posiciones.

Parámetros:

destino Referencia al arreglo destino

5.6.3.2. void copiar (const Gen & origen) [inline, virtual] Asigna nuevos valores al gen copiando las propiedades de otro gen.

Parámetros:

origen Objeto del que se hace copia

Implementa Gen.

5.6.3.3. Gen* crearCopia () const [inline, virtual] Crea una copia exacta del gen.

Devuelve:

Apuntador a un nuevo objeto GenArreglo<G,T> idéntico al actual.

Implementa Gen.

5.6.3.4. G * crearGen () const [protected] Crea un nuevo objeto del tipo de genes contenidos en el arreglo y retorna su apuntador.

Si el objeto GenArreglo fue creado utilizando un valor inicial establecido, el nuevo gen se crea usando este mismo valor. De lo contrario se crea con el valor por defecto del tipo G.

Devuelve:

Apuntador a un objeto del tipo de genes del arreglo,

5.6.3.5. void generarAleatorio () [virtual] Genera un arreglo de genes de tamaño y valores aleatorios.

Asigna al gen un nuevo tamaño y valores aleatorios dentro de sus respectivos rangos.

Implementa Gen.

5.6.3.6. G& getGen (int pos) const [inline] Retorna el apuntador al gen indicado por *pos* en el arreglo de genes.

Parámetros:

pos Posición dentro del arreglo de genes

Devuelve:

Apuntador al [Gen](#) en la posición *pos*

5.6.3.7. const [GenEntero](#)* getGenItems () const [inline] Retorna el [GenEntero](#) que almacena el tamaño del arreglo variable.

5.6.3.8. int getMaxTam () const [inline] Retorna el tamaño máximo del arreglo variable.

5.6.3.9. T getMaxVal () const [inline] Retorna el valor máximo válido para los datos almacenados en el arreglo de genes.

5.6.3.10. int getMinTam () const [inline] Retorna el tamaño mínimo del arreglo variable.

5.6.3.11. T getMinVal () const [inline] Retorna el valor mínimo válido para los datos almacenados en el arreglo de genes.

5.6.3.12. int getTam () const [inline] Retorna el tamaño actual del arreglo de genes.

5.6.3.13. T getVal (int *pos*) const [inline] Retorna el valor actual del gen en la posición *pos* del arreglo de genes.

Parámetros:

pos Posición dentro del arreglo de genes

Devuelve:

Valor actual del gen en la posición *pos*

5.6.3.14. [OperadorCruce](#) * operadorCruceDefecto () const [virtual]
Retorna un objeto correspondiente al operador de cruce establecido por defecto para genes de tipo arreglo.

Devuelve:

Apuntador al nuevo objeto de la clase `OperadorCruceArreglo<G,T>`

Implementa [Gen](#).

5.6.3.15. [OperadorMutacion](#) * operadorMutacionDefecto () const [virtual] Retorna un objeto correspondiente al operador de mutación establecido por defecto para genes de tipo arreglo.

Devuelve:

Apuntador al nuevo objeto de la clase `OperadorMutacionArreglo<G,T>`

Implementa [Gen](#).

5.6.3.16. const [GenArreglo](#)< G, T > & operator= (const [Arreglo](#)< T > & *origen*) Operador de asignación a partir de un objeto `Arreglo<T>`.

Asigna nuevos valores al gen copiando los valores de un arreglo de datos de tipo T

Parámetros:

origen **Arreglo** del que se toman los valores

Devuelve:

Referencia al objeto actual.

5.6.3.17. const **GenArreglo< G, T > & operator= (const **GenArreglo**< G, T > & *origen*)** Operador de asignación a partir de otro objeto **GenArreglo**<G,T>.

Asigna nuevos valores al gen copiando las propiedades de otro gen del mismo tipo

Parámetros:

origen Objeto del que se hace copia

Devuelve:

Referencia al objeto actual.

5.6.3.18. int setTam (int *nuevoTam*, bool *crearAleatorios* = false) Cambia el tamaño del arreglo.

Si el nuevo tamaño es menor que el anterior, se eliminan los genes restantes. Si es mayor, los nuevos genes son creados aleatoriamente dependiendo del valor de *crearAleatorios*

Parámetros:

nuevoTam Nuevo tamaño del arreglo.

crearAleatorios Si su valor es true se crean genes aleatorios. Si es false se crean genes con valores iniciales preestablecidos

Devuelve:

Nuevo tamaño del arreglo.

5.6.3.19. `bool setVal (int pos, T valor)` Asigna un nuevo valor al gen ubicado en la posición *pos*.

Si *pos* no es una posición válida en el arreglo, no se asigna el valor especificado.

Parámetros:

pos Posición dentro del arreglo de genes

valor Valor a asignar al gen en la posición *pos*

Devuelve:

true si el valor especificado fue asignado, false en caso contrario.

5.6.4. Documentación de los datos miembro

5.6.4.1. `Arreglo<G>* m_pArregloGen` [protected] `Arreglo` que contiene la información genética, con genes del tipo especificado por *G*.

5.6.4.2. `GenEntero* m_pGenItems` [protected] Apuntador al `GenEntero` que almacena el tamaño actual del arreglo.

5.6.4.3. `bool m_usarValInicial` [protected] Indica si se debe usar el valor inicial de cada gen al crearlo

5.6.4.4. `T m_valInicial` [protected] Valor inicial para los datos contenidos en el arreglo.

5.6.4.5. `T m_valMax` [protected] Máximo valor válido para los datos contenidos en el arreglo.

5.6.4.6. T [m_valMin](#) `[protected]` Mínimo valor válido para los datos contenidos en el arreglo.

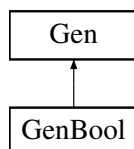
La documentación para esta clase fué generada a partir del siguiente archivo:

- [genarreglo.h](#)

5.7. Referencia de la Clase GenBool

```
#include <genbool.h>
```

Diagrama de herencias de GenBool:



5.7.1. Descripción detallada

Clase derivada de la clase [Gen](#) especializada en un gen de tipo booleano.

Se utiliza para representar y almacenar la información genética de variables cuya codificación se realiza con booleanos (0's y 1's).

Métodos públicos

- [GenBool](#) (const [GenBool](#) &origen)

Constructor por copia.

- [GenBool](#) (bool Min=false, bool Max=true)

Constructor por defecto.

- [GenBool](#) (bool Min, bool Max, bool ValorInicial)

Constructor con valor inicial.

- [Gen](#) * [crearCopia](#) () const

Crea una copia exacta del gen.

- void `copiar` (const `Gen` &origen)

Asigna nuevos valores al gen copiando las propiedades de otro gen.

- const `GenBool` & `operator=` (const bool &nuevoValor)

Operador de asignación a partir de un dato tipo bool.

- `operator const bool` () const

Conversión implícita a bool.

- `~GenBool` ()

Destructor.

- bool `setVal` (bool nuevoValor)

Ajusta el valor actual del dato almacenado por el gen. Retorna el valor asignado.

- bool `getVal` () const

Retorna el valor actual del dato almacenado en el gen.

- void `generarAleatorio` ()

Genera un valor aleatorio para el dato almacenado en el gen.

- `OperadorMutacion` * `operadorMutacionDefecto` () const

Retorna un objeto correspondiente al operador de mutación establecido por defecto para genes booleanos.

- `OperadorCruce` * `operadorCruceDefecto` () const

Retorna un objeto correspondiente al operador de cruce establecido por defecto para genes booleanos.

Atributos protegidos

- bool `m_Valor`

Almacena la información del gen.

5.7.2. Documentación del constructor y destructor

5.7.2.1. `GenBool` (const `GenBool` & *origen*) `[inline]` Constructor por copia.

Construye el nuevo objeto copiando idénticamente las propiedades de otro objeto de la clase `GenBool`.

Parámetros:

origen Objeto del que se hace copia.

5.7.2.2. `GenBool` (bool *Min* = false, bool *Max* = true) `[inline]` Constructor por defecto.

Inicializa el dato contenido por el gen en un valor aleatorio

Parámetros:

Min Es ignorado. Existe por compatibilidad.

Max Es ignorado. Existe por compatibilidad.

5.7.2.3. `GenBool` (bool *Min*, bool *Max*, bool *ValorInicial*) `[inline]` Constructor con valor inicial.

Inicializa el valor del gen con un valor suministrado por el usuario.

Parámetros:

Min Es ignorado. Existe por compatibilidad.

Max Es ignorado. Existe por compatibilidad.

ValorInicial Valor inicial del gen.

5.7.2.4. `~GenBool () [inline]` Destructor.

5.7.3. Documentación de las funciones miembro

5.7.3.1. `void copiar (const Gen & origen) [inline, virtual]` Asigna nuevos valores al gen copiando las propiedades de otro gen.

Parámetros:

origen Objeto del que se hace copia

Implementa `Gen`.

5.7.3.2. `Gen* crearCopia () const [inline, virtual]` Crea una copia exacta del gen.

Devuelve:

Apuntador a un nuevo objeto GenBool idéntico al actual.

Implementa `Gen`.

5.7.3.3. `void generarAleatorio () [inline, virtual]` Genera un valor aleatorio para el dato almacenado en el gen.

Implementa `Gen`.

5.7.3.4. `bool getVal () const [inline]` Retorna el valor actual del dato almacenado en el gen.

Devuelve:

Valor actual del gen

5.7.3.5. `OperadorCruce * operadorCruceDefecto () const [inline, virtual]` Retorna un objeto correspondiente al operador de cruce establecido por defecto para genes booleanos.

Crea un nuevo objeto de la clase [OperadorCruceBoolDiscreto](#), correspondiente al operador de cruce establecido por defecto para genes de tipo bool.

Devuelve:

Apuntador al nuevo objeto de la clase [OperadorCruceBoolPlano](#)

Implementa [Gen](#).

5.7.3.6. `OperadorMutacion * operadorMutacionDefecto () const [inline, virtual]` Retorna un objeto correspondiente al operador de mutación establecido por defecto para genes booleanos.

Crea un nuevo objeto de la clase [OperadorMutacionBoolUniforme](#), correspondiente al operador de mutación establecido por defecto para genes de tipo bool.

Devuelve:

Apuntador al nuevo objeto de la clase [OperadorMutacionBoolUniforme](#)

Implementa [Gen](#).

5.7.3.7. `operator const bool () const [inline]` Conversión implícita a bool.

Convierte un objeto GenBool en un dato de tipo bool utilizando su valor actual.
Permite hacer conversión explícita del estilo *(bool)gen1*

Devuelve:

Valor actual del gen.

5.7.3.8. const [GenBool](#)& operator= (const bool & *nuevoValor*) [inline]

Operador de asignación a partir de un dato tipo bool.

Asigna al gen el valor de un dato de tipo bool.

Parámetros:

nuevoValor Booleano que se asigna al gen

Devuelve:

Referencia al gen.

5.7.3.9. bool setVal (bool *nuevoValor*) [inline] Ajusta el valor actual del dato almacenado por el gen. Retorna el valor asignado.

Parámetros:

nuevoValor Valor a asignar al gen.

Devuelve:

Valor asignado.

5.7.4. Documentación de los datos miembro

5.7.4.1. bool [m_Valor](#) [protected] Almacena la información del gen.

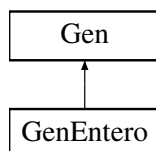
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genbool.h](#)
- [genbool.cpp](#)

5.8. Referencia de la Clase GenEntero

```
#include <genentero.h>
```

Diagrama de herencias de GenEntero:



5.8.1. Descripción detallada

Clase derivada de la clase [Gen](#) especializada en un gen de tipo entero.

Se utiliza para representar y almacenar la información genética de variables cuya codificación se realiza con números enteros.

Métodos públicos

- [GenEntero](#) (const [GenEntero](#) &origen)

Constructor por copia.

- [GenEntero](#) (long Min=-10, long Max=10)

Constructor por defecto.

- [GenEntero](#) (long Min, long Max, long ValorInicial)

Constructor con valor inicial.

- [Gen](#) * [crearCopia](#) () const

Crea una copia exacta del gen.

- void `copiar` (const `Gen` &origen)

Asigna nuevos valores al gen copiando las propiedades de otro gen.

- const `GenEntero` & `operator=` (const long &nuevoValor)

Operador de asignación a partir de un dato entero.

- `operator const long` () const

Conversión implícita a long.

- `~GenEntero` ()

Destructor.

- long `setVal` (long nuevoValor)

Ajusta el valor actual del dato almacenado por el gen. Retorna el valor asignado.

- long `getVal` () const

Retorna el valor actual del dato almacenado en el gen.

- long `getMax` () const

Retorna el valor máximo que puede tomar el gen.

- long `getMin` () const

Retorna el valor mínimo que puede tomar el gen.

- void `generarAleatorio` ()

Genera un valor aleatorio para el dato almacenado en el gen.

- `OperadorMutacion` * `operadorMutacionDefecto` () const

Retorna un objeto correspondiente al operador de mutación establecido por defecto para genes enteros.

- `OperadorCruce * operadorCruceDefecto () const`

Retorna un objeto correspondiente al operador de cruce establecido por defecto para genes enteros.

Atributos protegidos

- `long m_Valor`

Almacena la información del gen.

- `long m_Minimo`

Mínimo valor que puede tomar el gen. Solo puede modificarse en el constructor.

- `long m_Maximo`

Máximo valor que puede tomar el gen. Solo puede modificarse en el constructor.

5.8.2. Documentación del constructor y destructor

5.8.2.1. `GenEntero (const GenEntero & origen) [inline]` Constructor por copia.

Construye el nuevo objeto copiando idénticamente las propiedades de otro objeto de la clase `GenEntero`.

Parámetros:

origen Objeto del que se hace copia.

5.8.2.2. `GenEntero` (long *Min* = -10, long *Max* = 10) Constructor por defecto.

Inicializa todos los miembros de la clase. Da al gen un valor por defecto promediando los valores *Min* y *Max*.

Parámetros:

Min Valor mínimo que puede tomar el gen, Por defecto es -10.

Max Valor máximo que puede tomar el gen, Por defecto es 10.

5.8.2.3. `GenEntero` (long *Min*, long *Max*, long *ValorInicial*) Constructor con valor inicial.

Inicializa todos los miembros de la clase con valores suministrados por el usuario.

Parámetros:

Min Valor mínimo que puede tomar el gen,

Max Valor máximo que puede tomar el gen, Si es menor que *Min*, se hace igual a *Min*.

ValorInicial Valor inicial del gen. Se restringe al rango [*Min*, *Max*]

5.8.2.4. `~GenEntero` () [inline] Destructor.

5.8.3. Documentación de las funciones miembro

5.8.3.1. `void copiar (const Gen & origen)` [inline, virtual] Asigna nuevos valores al gen copiando las propiedades de otro gen.

Parámetros:

origen Objeto del que se hace copia

Implementa `Gen`.

5.8.3.2. `Gen* crearCopia () const [inline, virtual]` Crea una copia exacta del gen.

Devuelve:

Apuntador a un nuevo objeto `GenEntero` idéntico al actual.

Implementa `Gen`.

5.8.3.3. `void generarAleatorio () [inline, virtual]` Genera un valor aleatorio para el dato almacenado en el gen.

Asigna al gen un nuevo valor entero aleatorio en el rango establecido

Implementa `Gen`.

5.8.3.4. `long getMax () const [inline]` Retorna el valor máximo que puede tomar el gen.

Devuelve:

Valor máximo permitido para el gen.

5.8.3.5. `long getMin () const [inline]` Retorna el valor mínimo que puede tomar el gen.

Devuelve:

Valor mínimo permitido para el gen.

5.8.3.6. `long getVal () const [inline]` Retorna el valor actual del dato almacenado en el gen.

Devuelve:

Valor actual del gen

5.8.3.7. [OperadorCruce](#) * operadorCruceDefecto () const `[inline, virtual]` Retorna un objeto correspondiente al operador de cruce establecido por defecto para genes enteros.

Crea un nuevo objeto de la clase [OperadorCruceEnteroBLX](#), correspondiente al operador de cruce establecido por defecto para genes enteros.

Devuelve:

Apuntador al nuevo objeto de la clase [OperadorCruceEnteroBLX](#)

Implementa [Gen](#).

5.8.3.8. [OperadorMutacion](#) * operadorMutacionDefecto () const `[inline, virtual]` Retorna un objeto correspondiente al operador de mutación establecido por defecto para genes enteros.

Crea un nuevo objeto de la clase [OperadorMutacionEnteroUniforme](#) correspondiente al operador de mutación establecido por defecto para genes enteros.

Devuelve:

Apuntador al nuevo objeto de la clase [OperadorMutacionEnteroUniforme](#)

Implementa [Gen](#).

5.8.3.9. `operator const long () const` `[inline]` Conversión implícita a long.

Convierte un objeto GenEntero en un dato de tipo long utilizando su valor actual. Permite hacer operaciones del estilo $(long)gen1 + (long)gen2$.

Devuelve:

Valor actual del gen.

5.8.3.10. `const GenEntero& operator= (const long & nuevoValor)`
[inline] Operador de asignación a partir de un dato entero.

Asigna al gen el valor de un dato de tipo entero.

Parámetros:

nuevoValor Número entero que se asigna al gen

Devuelve:

Referencia al gen.

5.8.3.11. `long setVal (long nuevoValor)` [inline] Ajusta el valor actual del dato almacenado por el gen. Retorna el valor asignado.

Parámetros:

nuevoValor Valor a asignar al gen. Es restringido al rango [*m_Minimo*, *m_Maximo*].

Devuelve:

Valor asignado.

5.8.4. Documentación de los datos miembro

5.8.4.1. `long m_Maximo` [protected] Máximo valor que puede tomar el gen. Solo puede modificarse en el constructor.

5.8.4.2. `long m_Minimo` [protected] Mínimo valor que puede tomar el gen. Solo puede modificarse en el constructor.

5.8.4.3. `long m_Valor` [protected] Almacena la información del gen.

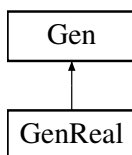
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genentero.h](#)
- [genentero.cpp](#)

5.9. Referencia de la Clase GenReal

```
#include <genreal.h>
```

Diagrama de herencias de GenReal:



5.9.1. Descripción detallada

Clase derivada de la clase [Gen](#) especializada en un gen de tipo real.

Se utiliza para representar y almacenar la información genética de variables cuya codificación se realiza con números reales.

Métodos públicos

- [GenReal](#) (const [GenReal](#) &origen)
Constructor por copia.
- [GenReal](#) (double Min=0.0, double Max=1.0)
Constructor por defecto.
- [GenReal](#) (double Min, double Max, double ValorInicial)
Constructor con valor inicial.
- [Gen](#) * [crearCopia](#) () const
Crea una copia exacta del gen.

- void `copiar` (const `Gen` &origen)
Asigna nuevos valores al gen copiando las propiedades de otro gen.
- const `GenReal` & `operator=` (const double &nuevoValor)
Operador de asignación a partir de un dato real.
- `operator const double` () const
Conversión implícita a double.
- `~GenReal` ()
Destructor.
- double `setVal` (double nuevoValor)
Ajusta el valor actual del dato almacenado por el gen. Retorna el valor asignado.
- double `getVal` () const
Retorna el valor actual del dato almacenado en el gen.
- double `getMax` () const
Retorna el valor máximo que puede tomar el gen.
- double `getMin` () const
Retorna el valor mínimo que puede tomar el gen.
- void `generarAleatorio` ()
Genera un valor aleatorio para el dato almacenado en el gen.
- `OperadorMutacion` * `operadorMutacionDefecto` () const

Retorna un objeto correspondiente al operador de mutación establecido por defecto para genes enteros.

- `OperadorCruce * operadorCruceDefecto () const`

Retorna un objeto correspondiente al operador de cruce establecido por defecto para genes enteros.

Atributos protegidos

- `double m_Valor`

Almacena la Información del gen.

- `double m_Minimo`

Mínimo valor que puede tomar el gen. Solo puede modificarse en el constructor.

- `double m_Maximo`

Máximo valor que puede tomar el gen. Solo puede modificarse en el constructor.

5.9.2. Documentación del constructor y destructor

5.9.2.1. `GenReal (const GenReal & origen) [inline]` Constructor por copia.

Construye el nuevo objeto copiando idénticamente las propiedades de otro objeto de la clase `GenReal`.

Parámetros:

origen Objeto del que se hace copia.

5.9.2.2. GenReal (double *Min* = 0.0, double *Max* = 1.0) Constructor por defecto.

Inicializa todos los miembros de la clase. Da al gen un valor por defecto promediando los valores *Min* y *Max*.

Parámetros:

Min Valor mínimo que puede tomar el gen, Por defecto es 0.0.

Max Valor máximo que puede tomar el gen, Por defecto es 1.0.

5.9.2.3. GenReal (double *Min*, double *Max*, double *ValorInicial*) Constructor con valor inicial.

Inicializa todos los miembros de la clase con valores suministrados por el usuario.

Parámetros:

Min Valor mínimo que puede tomar el gen,

Max Valor máximo que puede tomar el gen, Si es menor que *Min*, se hace igual a *Min*.

ValorInicial Valor inicial del gen. Se restringe al rango [*Min*, *Max*]

5.9.2.4. ~GenReal () [inline] Destructor.

5.9.3. Documentación de las funciones miembro

5.9.3.1. void copiar (const Gen & origen) [inline, virtual] Asigna nuevos valores al gen copiando las propiedades de otro gen.

Parámetros:

origen Objeto del que se hace copia

Implementa Gen.

5.9.3.2. [Gen](#)* crearCopia () const [inline, virtual] Crea una copia exacta del gen.

Devuelve:

Apuntador a un nuevo objeto GenReal idéntico al actual.

Implementa [Gen](#).

5.9.3.3. void generarAleatorio () [inline, virtual] Genera un valor aleatorio para el dato almacenado en el gen.

Asigna al gen un nuevo valor real aleatorio en el rango establecido

Implementa [Gen](#).

5.9.3.4. double getMax () const [inline] Retorna el valor máximo que puede tomar el gen.

Devuelve:

Valor máximo permitido para el gen.

5.9.3.5. double getMin () const [inline] Retorna el valor mínimo que puede tomar el gen.

Devuelve:

Valor mínimo permitido para el gen.

5.9.3.6. double getVal () const [inline] Retorna el valor actual del dato almacenado en el gen.

Devuelve:

Valor actual del gen

5.9.3.7. [OperadorCruce](#) * operadorCruceDefecto () const `[inline, virtual]` Retorna un objeto correspondiente al operador de cruce establecido por defecto para genes enteros.

Crea un nuevo objeto de la clase [OperadorCruceRealBLX](#), correspondiente al operador de cruce establecido por defecto para genes reales.

Devuelve:

Apuntador al nuevo objeto de la clase [OperadorCruceRealBLX](#)

Implementa [Gen](#).

5.9.3.8. [OperadorMutacion](#) * operadorMutacionDefecto () const `[inline, virtual]` Retorna un objeto correspondiente al operador de mutación establecido por defecto para genes enteros.

Crea un nuevo objeto de la clase [OperadorMutacionRealUniforme](#), correspondiente al operador de mutación establecido por defecto para genes reales.

Devuelve:

Apuntador al nuevo objeto de la clase [OperadorMutacionRealUniforme](#)

Implementa [Gen](#).

5.9.3.9. `operator const double () const` `[inline]` Conversión implícita a double.

Convierte un objeto GenReal en un dato de tipo double utilizando su valor actual. Permite hacer operaciones del estilo $(double)gen1 + (double)gen2$.

Devuelve:

Valor actual del gen.

5.9.3.10. const `GenReal`& operator= (const double & *nuevoValor*)
[inline] Operador de asignación a partir de un dato real.

Asigna al gen el valor de un dato de tipo real.

Parámetros:

nuevoValor Número real que se asigna al gen

Devuelve:

Referencia al gen.

5.9.3.11. double setVal (double *nuevoValor*) [inline] Ajusta el valor actual del dato almacenado por el gen. Retorna el valor asignado.

Parámetros:

nuevoValor Valor a asignar al gen. Es restringido al rango [m_Minimo, m_Maximo].

Devuelve:

Valor asignado.

5.9.4. Documentación de los datos miembro

5.9.4.1. double `m_Maximo` [protected] Máximo valor que puede tomar el gen. Solo puede modificarse en el constructor.

5.9.4.2. double `m_Minimo` [protected] Mínimo valor que puede tomar el gen. Solo puede modificarse en el constructor.

5.9.4.3. double `m_Valor` [protected] Almacena la Información del gen.

La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genreal.h](#)
- [genreal.cpp](#)

5.10. Referencia de la Clase Individuo

```
#include <genetico.h>
```

5.10.1. Descripción detallada

Clase que administra la información genética de un individuo.

Un objeto de la clase Individuo está conformado por un arreglo de apuntadores a objetos de clases derivadas de [Gen](#). Además cuenta con funciones que le proporcionan las características básicas necesarias para el algoritmo genético.

Métodos públicos

- [Individuo](#) ([AlgoritmoGenetico](#) *pAG)

Constructor por defecto.

- [Individuo](#) (const [Individuo](#) &origen)

Constructor por copia.

- const [Individuo](#) & [operator=](#) (const [Individuo](#) &origen)

- virtual ~[Individuo](#) ()

Destructor virtual.

- int [getTamGenoma](#) () const

Retorna el número de genes presentes en el genoma del individuo.

- const [Gen](#) & [getGen](#) (int pos) const

Devuelve el gen ubicado en una posición determinada del genoma.

- int **adicionarGen** (Gen *pGen)
Adiciona un gen al final del genoma del individuo.
- Gen * **reemplazarGen** (Gen *pNuevoGen, int pos)
Reemplaza el gen ubicado en una posición determinada del genoma por otro gen.
- void **SetAG** (AlgoritmoGenetico *pAG)
Cambia el algoritmo genetico al que pertenece el individuo.
- void **asignarProbabilidad** (double probabilidad)
Asigna la probabilidad de supervivencia del individuo.
- void **generarAleatorio** ()
Ordena al individuo que se genere aleatoriamente.
- double **objetivo** (bool actualizarAG=false)
- void **mutar** ()
Ordena al individuo que pase por el proceso de mutación.
- double **getProbabilidad** () const
Retorna la probabilidad de supervivencia del individuo.
- Individuo * **getPareja** () const
Retorna un apuntador al individuo con el que se efectúa el cruce.
- void **asignarPareja** (Individuo *pPareja)
Asigna el apuntador al individuo con el que efectuará el cruce.

Atributos protegidos

- `Arreglo< Gen > * m_pGenoma`

Arreglo de apuntadores a objetos de las clases derivadas de `Gen` que contienen la información genética del individuo.

- `AlgoritmoGenetico * m_pAG`

Apuntador al algoritmo genetico al que pertenece el individuo.

- `double m_Probabilidad`

Almacena la probabilidad de supervivencia del individuo.

- `Individuo * m_pPareja`

Apuntador a la pareja con la que el individuo efectuará el cruce de sus genes.

Atributos privados

- `bool m_objetivoActualizado`

Indica si `m_Objetivo` refleja el valor actual de la función objetivo.

- `double m_Objetivo`

Almacena el valor de la función objetivo.

5.10.2. Documentación del constructor y destructor

5.10.2.1. **Individuo** (**AlgoritmoGenetico** * **pAG**) Constructor por defecto.

Inicializa las propiedades básicas del individuo

Parámetros:

pAG Apuntador al objeto **AlgoritmoGenetico** al que pertenece.

5.10.2.2. **Individuo** (const **Individuo** & *origen*) Constructor por copia.

Construye el nuevo objeto copiando idénticamente las propiedades de otro objeto de la clase Individuo.

Parámetros:

origen Objeto del que se hace copia.

5.10.2.3. ~**Individuo** () [virtual] Destructor virtual.

Destruye todos sus componentes

5.10.3. Documentación de las funciones miembro

5.10.3.1. **int** **adicionarGen** (**Gen** * **pGen**) [inline] Adiciona un gen al final del genoma del individuo.

Parámetros:

pGen Aputador al gen a adicionar al genoma.

Devuelve:

Posición en la que fue adicionado el gen.

5.10.3.2. void asignarPareja ([Individuo](#) * *pPareja*) [inline] Asigna el apuntador al individuo con el que efectuará el cruce.

Parámetros:

pPareja Apuntador al individuo con el que efectuará el cruce.

5.10.3.3. void asignarProbabilidad (double *probabilidad*) [inline] Asigna la probabilidad de supervivencia del individuo.

Parámetros:

probabilidad Nueva probabilidad de supervivencia

5.10.3.4. void generarAleatorio () Ordena al individuo que se genere aleatoriamente.

Hace que cada uno de los genes presentes en el genoma se genere aleatoriamente invocando su miembro [generarAleatorio\(\)](#).

5.10.3.5. const [Gen&](#) getGen (int *pos*) const [inline] Devuelve el gen ubicado en una posición determinada del genoma.

Parámetros:

pos Posición del gen a obtener

Devuelve:

referencia al gen ubicado en la posición *pos*. Este no podrá ser modificado.

5.10.3.6. [Individuo*](#) getPareja () const [inline] Retorna un apuntador al individuo con el que se efectúa el cruce.

Devuelve:

Apuntador al individuo con el que se efectúa el cruce

5.10.3.7. double getProbabilidad () const [inline] Retorna la probabilidad de supervivencia del individuo.

Devuelve:

Probabilidad de supervivencia del individuo

5.10.3.8. int getTamGenoma () const [inline] Retorna el número de genes presentes en el genoma del individuo.

return Tamaño del genoma

5.10.3.9. void mutar () Ordena al individuo que pase por el proceso de mutación.

Hace que cada uno de los genes presentes en el individuo pase por el operador de mutación respectivo

5.10.3.10. double objetivo (bool actualizarAG = false) Retorna el valor de la funcion objetivo del individuo

5.10.3.11. const Individuo & operator= (const Individuo & origen) Copia idénticamente las propiedades de otro objeto de la clase Individuo.

Parámetros:

origen Objeto del que se hace copia

Devuelve:

Referencia al individuo

5.10.3.12. `Gen*` `reemplazarGen` (`Gen` * `pNuevoGen`, `int pos`) `[inline]`

Reemplaza el gen ubicado en una posición determinada del genoma por otro gen.

Parámetros:

pNuevoGen Apuntador al gen que reemplazará al gen ubicado en la posición especificada.

pos Posición del gen a reemplazar.

Devuelve:

Apuntador al gen reemplazado.

5.10.3.13. `void SetAG` (`AlgoritmoGenetico` * `pAG`) `[inline]` Cambia el algoritmo genetico al que pertenece el individuo.

Parámetros:

pAG Apuntador al objeto AlgoritmoGenético del que hará parte el individuo.

5.10.4. Documentación de los datos miembro

5.10.4.1. `double m_Objetivo` `[private]` Almacena el valor de la función objetivo.

5.10.4.2. `bool m_objetivoActualizado` `[private]` Indica si *m_Objetivo* refleja el valor actual de la función objetivo.

Se establece a *false* después de cualquier cambio en el genoma para recalcular la funcion objetivo cuando es invocado el método *objetivo()*.

5.10.4.3. `AlgoritmoGenetico*` `m_pAG` `[protected]` Apuntador al algoritmo genetico al que pertenece el individuo.

5.10.4.4. [Arreglo<Gen>*](#) [m_pGenoma](#) [protected] [Arreglo](#) de apun-
dores a objetos de las clases derivadas de [Gen](#) que contienen la información
genética del individuo.

5.10.4.5. [Individuo*](#) [m_pPareja](#) [protected] Apuntador a la pareja con la
que el individuo efectuará el cruce de sus genes.

5.10.4.6. `double` [m_Probabilidad](#) [protected] Almacena la probabilidad
de supervivencia del individuo.

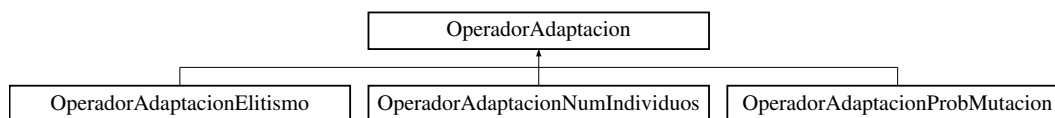
La documentación para esta clase fué generada a partir de los siguientes archi-
vos:

- [genetico.h](#)
- [genetico.cpp](#)

5.11. Referencia de la Clase OperadorAdaptacion

```
#include <genetico.h>
```

Diagrama de herencias de OperadorAdaptacion:



5.11.1. Descripción detallada

Clase abstracta que define el proceso de adaptación del algoritmo genético.

El proceso de adaptación hace que se modifiquen los parámetros del algoritmo genético durante su ejecución. Un objeto de una clase derivada de OperadorAdaptacion opera sobre el algoritmo genético realizando cambios en su estructura funcional con el fin de variar su desempeño.

Métodos públicos

- **OperadorAdaptacion ()**

Constructor por defecto.

- **virtual ~OperadorAdaptacion ()**

Destructor.

- **virtual void adaptacion (AlgoritmoGenetico *pAG)=0**

Ejecuta el procedimiento de adaptacion.

5.11.2. Documentación del constructor y destructor

5.11.2.1. **OperadorAdaptacion ()** [inline] Constructor por defecto.

Debe sobrecargarse en las clases derivadas

5.11.2.2. **virtual ~OperadorAdaptacion ()** [inline, virtual] Destructor.

Es virtual para poder sobrecargarse en las clases derivadas

5.11.3. Documentación de las funciones miembro

5.11.3.1. **virtual void adaptacion (AlgoritmoGenetico * pAG)** [pure virtual] Ejecuta el procedimiento de adaptacion.

Debe sobrecargarse en las clases derivadas

Parámetros:

pAG Apuntador al algoritmo genético sobre el que opera

Implementado en [OperadorAdaptacionElitismo](#), [OperadorAdaptacionProbMutacion](#), y [OperadorAdaptacionNumIndividuos](#).

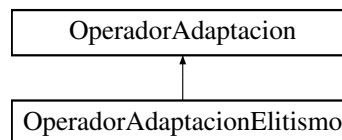
La documentación para esta clase fué generada a partir del siguiente archivo:

- [genetico.h](#)

5.12. Referencia de la Clase OperadorAdaptacion-Elitismo

```
#include <genetico.h>
```

Diagrama de herencias de OperadorAdaptacionElitismo:



5.12.1. Descripción detallada

Clase derivada de la clase [OperadorAdaptacion](#) encargada de efectuar el proceso de elitismo para el algoritmo.

El elitismo consiste en obtener el mejor individuo de la generación actual y copiarlo en la siguiente generación.

Métodos públicos

- [OperadorAdaptacionElitismo](#) ()

Constructor.

- [~OperadorAdaptacionElitismo](#) ()

Destructor.

- void [adaptacion](#) ([AlgoritmoGenetico](#) *pAG)

Método que efectúa el proceso de elitismo.

5.12.2. Documentación del constructor y destructor

5.12.2.1. [OperadorAdaptacionElitismo \(\)](#) [inline] Constructor.

5.12.2.2. [~OperadorAdaptacionElitismo \(\)](#) [inline] Destructor.

5.12.3. Documentación de las funciones miembro

5.12.3.1. **void adaptacion ([AlgoritmoGenetico](#) * *pAG*)** [virtual] Método que efectúa el proceso de elitismo.

Parámetros:

pAG Apuntador al algoritmo genético sobre el cual opera.

Implementa [OperadorAdaptacion](#).

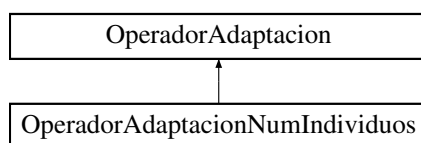
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genetico.h](#)
- [genetico.cpp](#)

5.13. Referencia de la Clase OperadorAdaptacion-NumIndividuos

```
#include <genetico.h>
```

Diagrama de herencias de OperadorAdaptacionNumIndividuos:



5.13.1. Descripción detallada

Clase derivada de la clase [OperadorAdaptacion](#) que hace una variación del número de individuos del algoritmo genético.

La primera generación del algoritmo contiene un número máximo de individuos establecido, este número disminuye linealmente hasta alcanzar un número mínimo de individuos predefinido. Este límite se alcanza cuando se llega a la última generación establecida para el algoritmo.

Métodos públicos

- [OperadorAdaptacionNumIndividuos](#) ([AlgoritmoGenetico](#) *pAG, int NumIndivInicio=0, int NumIndivFin=0)

Constructor.

- [~OperadorAdaptacionNumIndividuos](#) ()

Destructor.

- void **adaptacion** (**AlgoritmoGenetico** *pAG)

Método para efectuar el proceso de adaptación de número de individuos.

Atributos protegidos

- int **m_nIndivInicio**

Indica el número inicial de individuos en la población.

- int **m_nIndivFin**

Indica el número final de individuos en la población.

5.13.2. Documentación del constructor y destructor

5.13.2.1. OperadorAdaptacionNumIndividuos (**AlgoritmoGenetico** * **pAG**,
int NumIndivInicio = 0, **int NumIndivFin = 0**) Constructor.

Inicializa los tamaños inicial y final de la población para el algoritmo

Parámetros:

pAG Apuntador al algoritmo genético sobre el que opera.

NumIndivInicio Número inicial de individuos en la población, debe ser positivo. Por defecto es igual al número de individuos del algoritmo genético.

NumIndivFin Número final de individuos en la población, Por defecto es igual a la cuarta parte del número inicial de individuos del algoritmo genético.

5.13.2.2. ~OperadorAdaptacionNumIndividuos () [**inline**] Destructor.

5.13.3. Documentación de las funciones miembro

5.13.3.1. void adaptacion ([AlgoritmoGenetico](#) * *pAG*) [virtual] Método para efectuar el proceso de adaptación de número de individuos.

Implementa [OperadorAdaptacion](#).

5.13.4. Documentación de los datos miembro

5.13.4.1. int [m_nIndivFin](#) [protected] Indica el número final de individuos en la población.

5.13.4.2. int [m_nIndivInicio](#) [protected] Indica el número inicial de individuos en la población.

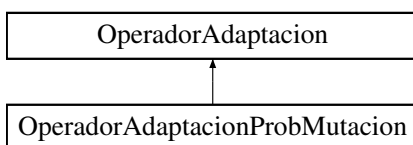
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genetico.h](#)
- [genetico.cpp](#)

5.14. Referencia de la Clase OperadorAdaptacion-ProbMutacion

```
#include <genetico.h>
```

Diagrama de herencias de OperadorAdaptacionProbMutacion:



5.14.1. Descripción detallada

Clase derivada de la clase [OperadorAdaptacion](#) que define la estrategia de adaptación para la probabilidad de mutación de los genes de cada individuo de la población.

Existen dos métodos de adaptación que varían la probabilidad de mutación:

- Variación escalonada: aumenta la probabilidad de mutación de forma gradual y escalonada para los genes cuando la variación de la medida Offline permanece en un rango establecido durante un número determinado de generaciones consecutivas.
- Variación exponencial: varía la probabilidad de mutación de todos los genes siguiendo un comportamiento exponencial decreciente determinado por $Pm(t) = P_{max}e^{-\frac{t}{T}}$ donde P_{max} es el valor máximo que puede tomar la probabilidad de mutación de un gen, t corresponde al número de la generación actual y T es un valor positivo preestablecido.

Métodos públicos

- `OperadorAdaptacionProbMutacion` (`AlgoritmoGenetico *pAG`, `int TipoAdaptacion=ADAPTACION_PROBMUTACION_EXPONENCIAL`)

Constructor.

- `~OperadorAdaptacionProbMutacion` ()

Destructor.

- `void adaptacion` (`AlgoritmoGenetico *pAG`)

Efectúa el proceso de adaptacion de variación de la probabilidad de mutación.

- `void setParamsOffline` (`double MaxProb=0.5`, `double FactorVariacion=0.001`, `double Escalon=0.1`, `int MaxCont=10`)

Cambia los parámetros para la ADAPTACION_PROBMUTACION_OFFLINE.

- `void setParamsExponencial` (`double MaxProb=0.5`, `double T=-1`)

Cambia los parámetros para la ADAPTACION_PROBMUTACION_EXPONENCIAL.

Atributos protegidos

- `int m_tipoAdaptacion`

Determina la forma de cambiar la probabilidad de mutación.

- `double m_maxProb`

Determina la máxima probabilidad de mutación en un gen.

- `int m_maxCont`

Determina el número máximo de generaciones que pueden pasar sin que varíe la probabilidad de mutación cuando la medida offline no varía en un factor mayor a m_factorVariacion. Sólo se utiliza en la ADAPTACION_PROBMUTACION_OFFLINE.

- `double m_factorVariacion`

Determina el factor de variación para la medida Offline. Sólo se utiliza en la ADAPTACION_PROBMUTACION_OFFLINE.

- `double m_escalon`

Determina la cantidad que se sumará a la probabilidad de mutación en cada variación. Sólo se utiliza en la ADAPTACION_PROBMUTACION_OFFLINE.

- `double m_T`

Valor positivo determina la atenuación de la curva exponencial. Sólo se utiliza en la ADAPTACION_PROBMUTACION_EXPONENCIAL.

5.14.2. Documentación del constructor y destructor

5.14.2.1. **OperadorAdaptacionProbMutacion** (**AlgoritmoGenetico** * *pAG*, `int TipoAdaptacion = ADAPTACION_PROBMUTACION_EXPONENCIAL`)
Constructor.

Inicializa el valor correspondiente al tipo de adaptación que será implementado e invoca a los métodos que establecen sus parámetros.

Parámetros:

pAG Apuntador al algoritmo genético sobre el que opera.

TipoAdaptacion Indica cuál tipo de adaptación para la probabilidad de mutación se va a implementar. Por defecto se implementa la ADAPTACION_PROBMUTACION_EXPONENCIAL.

5.14.2.2. `~OperadorAdaptacionProbMutacion ()` [inline] Destructor.

5.14.3. Documentación de las funciones miembro

5.14.3.1. `void adaptacion (AlgoritmoGenetico * pAG)` [virtual] Efectúa el proceso de adaptacion de variación de la probabilidad de mutación.

Implementa [OperadorAdaptacion](#).

5.14.3.2. `void setParamsExponencial (double MaxProb = 0.5, double T = -1)` Cambia los parámetros para la ADAPTACION_PROBMUTACION_EXPONENCIAL.

Parámetros:

MaxProb Valor máximo de probabilidad de mutación para asignar a un gen, se restringe al intervalo [0, 1]. Por defecto es igual a 0.5.

T Valor positivo determina la atenuación de la curva exponencial.

5.14.3.3. `void setParamsOffline (double MaxProb = 0.5, double FactorVariacion = 0.001, double Escalon = 0.1, int MaxCont = 10)` Cambia los parámetros para la ADAPTACION_PROBMUTACION_OFFLINE.

Parámetros:

MaxProb Valor máximo de probabilidad de mutación para asignar a un gen, se restringe al intervalo [0, 1]. Por defecto es igual a 0.5.

FactorVariacion Valor que determina el factor de variación permitido para la medida offline. Por defecto es igual a 0.001.

Escalon Valor en que se aumenta la probabilidad de mutación a un gen en cada variación. Por defecto es igual a 0.1.

MaxCont Número máximo de generaciones que pueden pasar sin que varíe la probabilidad de mutación cuando la medida offline no varía en un factor mayor a *m_factorVariacion*. Por defecto es igual a 10.

5.14.4. Documentación de los datos miembro

5.14.4.1. double *m_escalon* [protected] Determina la cantidad que se sumará a la probabilidad de mutación en cada variación. Sólo se utiliza en la ADAPTACION_PROB MUTACION_OFFLINE.

5.14.4.2. double *m_factorVariacion* [protected] Determina el factor de variación para la medida Offline. Sólo se utiliza en la ADAPTACION_PROB MUTACION_OFFLINE.

5.14.4.3. int *m_maxCont* [protected] Determina el número máximo de generaciones que pueden pasar sin que varíe la probabilidad de mutación cuando la medida offline no varía en un factor mayor a *m_factorVariacion*. Sólo se utiliza en la ADAPTACION_PROB MUTACION_OFFLINE.

5.14.4.4. double *m_maxProb* [protected] Determina la máxima probabilidad de mutación en un gen.

5.14.4.5. double *m_T* [protected] Valor positivo determina la atenuación de la curva exponencial. Sólo se utiliza en la ADAPTACION_PROB MUTACION_EXPONENCIAL.

5.14.4.6. int [m_tipoAdaptacion](#) `[protected]` Determina la forma de cambiar la probabilidad de mutación.

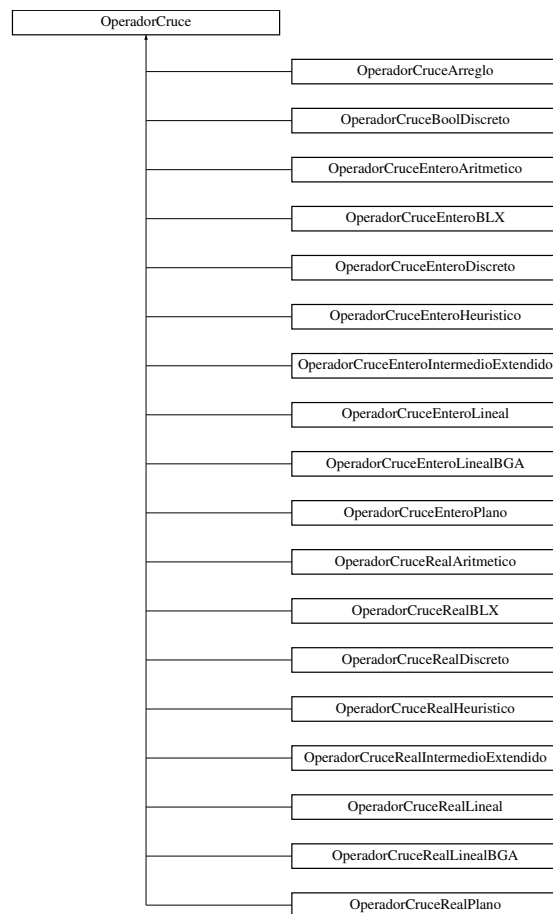
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genetico.h](#)
- [genetico.cpp](#)

5.15. Referencia de la Clase OperadorCruce

```
#include <genetico.h>
```

Diagrama de herencias de OperadorCruce:



5.15.1. Descripción detallada

Clase abstracta que administra el proceso de cruce entre los individuos del algoritmo genético.

El proceso de cruce consiste en intercambiar la información genética de dos individuos para generar nuevos individuos. Un objeto de una clase derivada de

OperadorCruce opera sobre objetos de clases derivadas de `Gen`, por lo tanto, estos objetos deben manejar el mismo tipo de dato.

Métodos públicos

- `OperadorCruce ()`

Constructor por defecto.

- `virtual ~OperadorCruce ()`

Destructor.

- `virtual void cruzarGenes (const Gen *pMejor, const Gen *pPeor, Arreglo< Gen > *pHijos, int numHijos, int indice)=0`

Método virtual que ejecuta el procedimiento de cruce entre dos individuos.

5.15.2. Documentación del constructor y destructor

5.15.2.1. `OperadorCruce ()` `[inline]` Constructor por defecto.

Debe sobrecargarse en las clases derivadas

5.15.2.2. `virtual ~OperadorCruce ()` `[inline, virtual]` Destructor.

Es virtual para definirse en las clases derivadas

5.15.3. Documentación de las funciones miembro

5.15.3.1. virtual void cruzarGenes (const [Gen](#) * *pMejor*, const [Gen](#) * *pPeor*, [Arreglo](#)< [Gen](#) > * *pHijos*, int *numHijos*, int *indice*) [pure virtual] Método virtual que ejecuta el procedimiento de cruce entre dos individuos.

Debe sobrecargarse en las clases derivadas.

Parámetros:

pMejor Apuntador al gen padre con la mejor función de evaluación.

pPeor Apuntador al gen padre con la peor función de evaluación.

pHijos [Arreglo](#) de genes al cual se adicionan los genes hijos.

numHijos Numero de genes hijos a crear en el cruce.

indice Valor que identifica al individuo de cuyos genes se realiza el cruce.

No es utilizado en todas las clases derivadas

Implementado en [OperadorCruceArreglo](#), [OperadorCruceBoolDiscreto](#), [OperadorCruceEnteroPlano](#), [OperadorCruceEnteroAritmetico](#), [OperadorCruceEnteroBLX](#), [OperadorCruceEnteroLineal](#), [OperadorCruceEnteroDiscreto](#), [OperadorCruceEnteroIntermedioExtendido](#), [OperadorCruceEnteroHeuristico](#), [OperadorCruceEnteroLinealBGA](#), [OperadorCruceRealPlano](#), [OperadorCruceRealAritmetico](#), [OperadorCruceRealBLX](#), [OperadorCruceRealLineal](#), [OperadorCruceRealDiscreto](#), [OperadorCruceRealIntermedioExtendido](#), [OperadorCruceRealHeuristico](#), y [OperadorCruceRealLinealBGA](#).

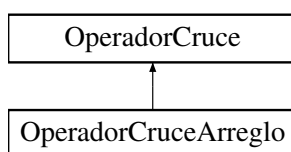
La documentación para esta clase fué generada a partir del siguiente archivo:

- [genetico.h](#)

5.16. Referencia de la Clase **OperadorCruceArreglo**

```
#include <genarreglo.h>
```

Diagrama de herencias de **OperadorCruceArreglo**:



5.16.1. Descripción detallada

template<class G, class T> class OperadorCruceArreglo< G, T > Clase derivada de la clase **OperadorCruce** usada en genes de tipo arreglo. G puede ser **GenBool**, **GenEntero** o **GenReal**. T puede ser bool, long o double.

Efectúa un cruce entre dos genes de tipo arreglo, creando nuevos genes hijos. El tamaño de los nuevos genes es definido por el cruce de los tamaños de los padres. Cada uno de los genes que hacen parte del nuevo arreglo, se obtiene cruzando los genes de la misma posición en el arreglo de los padres, usando el operador de cruce por defecto para genes de tipo G. Si el tamaño del nuevo gen es mayor que el tamaño del padre de menor tamaño, los elementos restantes se copian exactamente del padre de mayor tamaño. Si el tamaño del nuevo gen es aún mayor que el tamaño de ambos padres, los genes restantes se crean aleatoriamente.

Métodos públicos

- **OperadorCruceArreglo ()**

Constructor de la clase OperadorCruceArreglo.

- `~OperadorCruceArreglo ()`

Destructor de la clase OperadorCruceArreglo.

- `void cruzarGenes (const Gen *pMejor, const Gen *pPeor, Arreglo< Gen > *pHijos, int numHijos, int indice=-1)`

Define el proceso de cruce de los genes de tipo arreglo.

Atributos protegidos

- `OperadorCruce * m_pOperadorCruceGenes`

Apuntador al operador de cruce de los genes que hacen parte del arreglo.

- `OperadorCruce * m_pOperadorCruceTamanos`

Apuntador al operador de cruce para el tamaño del arreglo.

5.16.2. Documentación del constructor y destructor

5.16.2.1. `OperadorCruceArreglo ()` [inline] Constructor de la clase OperadorCruceArreglo.

5.16.2.2. `~OperadorCruceArreglo ()` [inline] Destructor de la clase OperadorCruceArreglo.

5.16.3. Documentación de las funciones miembro

5.16.3.1. `void cruzarGenes (const Gen * pMejor, const Gen * pPeor, Arreglo< Gen > * pHijos, int numHijos, int indice = -1) [virtual]` Define el proceso de cruce de los genes de tipo arreglo.

Parámetros:

pMejor [Gen](#) del individuo padre con mejor función de evaluación.

pPeor [Gen](#) del individuo padre con peor función de evaluación.

pHijos [Arreglo](#) de genes al cual se adicionan los genes hijos.

numHijos Numero de genes hijos a crear en el cruce.

indice Valor que identifica al individuo de cuyos genes se realiza el cruce.

Implementa [OperadorCruce](#).

5.16.4. Documentación de los datos miembro

5.16.4.1. [OperadorCruce](#)* [m_pOperadorCruceGenes](#) [protected]

Apuntador al operador de cruce de los genes que hacen parte del arreglo.

5.16.4.2. [OperadorCruce](#)* [m_pOperadorCruceTamanos](#) [protected]

Apuntador al operador de cruce para el tamaño del arreglo.

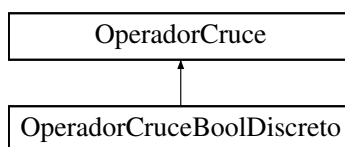
La documentación para esta clase fué generada a partir del siguiente archivo:

- [genarreglo.h](#)

5.17. Referencia de la Clase OperadorCruceBoolDiscreto

```
#include <genbool.h>
```

Diagrama de herencias de OperadorCruceBoolDiscreto:



5.17.1. Descripción detallada

Clase derivada de la clase [OperadorCruce](#) que define el cruce discreto entre dos genes de tipo booleano.

Crea *numHijos* genes cuyos valores corresponden a datos booleanos escogidos aleatoriamente entre los valores de *pMejor* y *pPeor*. Los valores resultantes se agregan al arreglo de genes *pHijos*.

Métodos públicos

- [OperadorCruceBoolDiscreto](#) ()
Constructor.
- [~OperadorCruceBoolDiscreto](#) ()
Destructor.
- void [cruzarGenes](#) (const [Gen](#) *pMejor, const [Gen](#) *pPeor, [Arreglo](#)< [Gen](#) > *pHijos, int numHijos, int indice=-1)

Ejecuta un cruce discreto sobre genes booleanos.

5.17.2. Documentación del constructor y destructor

5.17.2.1. **OperadorCruceBoolDiscreto ()** [inline] Constructor.

5.17.2.2. **~OperadorCruceBoolDiscreto ()** [inline] Destructor.

5.17.3. Documentación de las funciones miembro

5.17.3.1. **void cruzarGenes (const Gen * pMejor, const Gen * pPeor, Arreglo< Gen > * pHijos, int numHijos, int indice = -1)** [virtual] Ejecuta un cruce discreto sobre genes booleanos.

Parámetros:

pMejor Gen del individuo padre con mejor función de evaluación.

pPeor Gen del individuo padre con peor función de evaluación.

pHijos Arreglo de genes al cual se adicionan los genes hijos.

numHijos Numero de genes hijos a crear en el cruce.

indice Es ignorado. Existe por compatibilidad.

Implementa **OperadorCruce**.

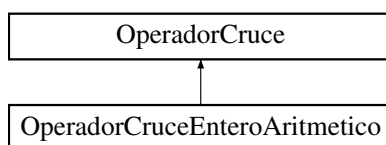
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genbool.h](#)
- [genbool.cpp](#)

5.18. Referencia de la Clase OperadorCruceEnteroAritmetico

```
#include <genentero.h>
```

Diagrama de herencias de OperadorCruceEnteroAritmetico:



5.18.1. Descripción detallada

Clase derivada de la clase [OperadorCruce](#) que define un cruce aritmético entre dos genes de tipo entero.

Crea *numHijos* genes cuyos valores corresponden a números enteros que se obtienen así:

$$h_1 = \lambda p_{Mejor} + (1 - \lambda) p_{Peor}$$

$$h_2 = \lambda p_{Peor} + (1 - \lambda) p_{Mejor}$$

Los valores resultantes se agregan al arreglo de genes *pHijos*

Métodos públicos

- [OperadorCruceEnteroAritmetico](#) (double Lambda=0.7)

Constructor.

- void [setLambda](#) (double Lambda)

Cambia el valor del parámetro Lambda comprobando los límites.

- double `getLambda ()`

Retorna el valor del parámetro Lambda.

- `~OperadorCruceEnteroAritmetico ()`

Destructor.

- void `cruzarGenes (const Gen *pMejor, const Gen *pPeor, Arreglo< Gen > *pHijos, int numHijos, int indice=-1)`

Ejecuta un cruce aritmético sobre genes enteros.

Atributos privados

- double `m_Lambda`

Parámetro que pondera cada uno de los genes padres.

5.18.2. Documentación del constructor y destructor

5.18.2.1. `OperadorCruceEnteroAritmetico` (double *Lambda* = 0.7)
`[inline]` Constructor.

Inicializa el valor del parámetro *Lambda*.

Parámetros:

Lambda Valor que pondera cada uno de los genes padres. Por defecto es igual a 0.7.

5.18.2.2. `~OperadorCruceEnteroAritmetico ()` `[inline]` Destructor.

5.18.3. Documentación de las funciones miembro

5.18.3.1. void cruzarGenes (const [Gen](#) * *pMejor*, const [Gen](#) * *pPeor*, [Arreglo](#)< [Gen](#) > * *pHijos*, int *numHijos*, int *indice* = -1) [virtual] Ejecuta un cruce aritmético sobre genes enteros.

Parámetros:

pMejor [Gen](#) del individuo padre con mejor función de evaluación.

pPeor [Gen](#) del individuo padre con peor función de evaluación.

pHijos [Arreglo](#) de genes al cual se adicionan los genes hijos.

numHijos Numero de genes hijos a crear en el cruce.

indice Es ignorado. Existe por compatibilidad.

Implementa [OperadorCruce](#).

5.18.3.2. double getLambda () [inline] Retorna el valor del parámetro *Lambda*.

5.18.3.3. void setLambda (double *Lambda*) [inline] Cambia el valor del parámetro *Lambda* comprobando los límites.

Parámetros:

Lambda Valor a asignar al parámetro *Lambda*.

5.18.4. Documentación de los datos miembro

5.18.4.1. double [m_Lambda](#) [private] Parámetro que pondera cada uno de los genes padres.

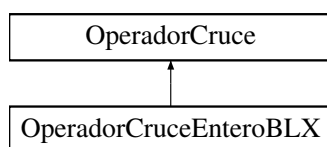
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genentero.h](#)
- [genentero.cpp](#)

5.19. Referencia de la Clase OperadorCruceEnteroBLX

```
#include <genentero.h>
```

Diagrama de herencias de OperadorCruceEnteroBLX:



5.19.1. Descripción detallada

Clase derivada de la clase [OperadorCruce](#) que define un cruce BLX $-\alpha$ entre dos genes de tipo entero.

Crea $numHijos$ genes cuyos valores corresponden a números enteros aleatorios del intervalo $[c_{min} - I\alpha, c_{max} + I\alpha]$ donde $c_{min} = \min[pMejor, pPeor]$, $c_{max} = \max[pMejor, pPeor]$ e $I = c_{max} - c_{min}$. Los valores resultantes se agregan al arreglo de genes $pHijos$

Métodos públicos

- [OperadorCruceEnteroBLX](#) (double Alfa=0.3)

Constructor.

- void [setAlfa](#) (double Alfa)

Cambia el valor del parámetro Alfa comprobando los límites.

- double [getAlfa](#) ()

Retorna el valor del parámetro Alfa.

- `~OperadorCruceEnteroBLX ()`

Destructor.

- `void cruzarGenes (const Gen *pMejor, const Gen *pPeor, Arreglo< Gen > *pHijos, int numHijos, int indice=-1)`

Ejecuta un cruce BLX - α sobre genes enteros.

Atributos privados

- `double m_Alfa`

Establece la amplitud del intervalo de definición.

5.19.2. Documentación del constructor y destructor

5.19.2.1. `OperadorCruceEnteroBLX (double Alfa = 0.3) [inline]` Constructor.

Inicializa Inicializa el valor de *Alfa*

Parámetros:

Alfa Establece la amplitud del intervalo de definición. Por defecto es igual a 0.3.

5.19.2.2. `~OperadorCruceEnteroBLX () [inline]` Destructor.

5.19.3. Documentación de las funciones miembro

5.19.3.1. void cruzarGenes (const [Gen](#) * *pMejor*, const [Gen](#) * *pPeor*, [Arreglo](#)< [Gen](#) > * *pHijos*, int *numHijos*, int *indice* = -1) [virtual] Ejecuta un cruce BLX - α sobre genes enteros.

Parámetros:

pMejor [Gen](#) del individuo padre con mejor función de evaluación.

pPeor [Gen](#) del individuo padre con peor función de evaluación.

pHijos [Arreglo](#) de genes al cual se adicionan los genes hijos.

numHijos Numero de genes hijos a crear en el cruce.

indice Es ignorado. Existe por compatibilidad.

Implementa [OperadorCruce](#).

5.19.3.2. double getAlfa () [inline] Retorna el valor del parámetro *Alfa*.

5.19.3.3. void setAlfa (double *Alfa*) [inline] Cambia el valor del parámetro *Alfa* comprobando los límites.

Parámetros:

Alfa Valor a asignar al parámetro *Alfa*

5.19.4. Documentación de los datos miembro

5.19.4.1. double [m_Alfa](#) [private] Establece la amplitud del intervalo de definición.

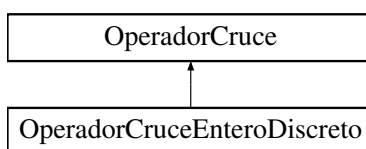
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genentero.h](#)
- [genentero.cpp](#)

5.20. Referencia de la Clase OperadorCruceEnteroDiscreto

```
#include <genentero.h>
```

Diagrama de herencias de OperadorCruceEnteroDiscreto:



5.20.1. Descripción detallada

Clase derivada de la clase `OperadorCruce` que define un cruce discreto entre dos genes de tipo entero.

Crea *numHijos* genes cuyos valores son tomados aleatoriamente del valor del gen padre o el del gen madre. Los genes resultantes se agregan al arreglo de genes *pHijos*

Métodos públicos

- `OperadorCruceEnteroDiscreto ()`
Constructor.
- `~OperadorCruceEnteroDiscreto ()`
Destructor.
- `void cruzarGenes (const Gen *pMejor, const Gen *pPeor, Arreglo< Gen > *pHijos, int numHijos, int indice=-1)`

Ejecuta un cruce discreto sobre genes enteros.

5.20.2. Documentación del constructor y destructor

5.20.2.1. **OperadorCruceEnteroDiscreto ()** [inline] Constructor.

5.20.2.2. **~OperadorCruceEnteroDiscreto ()** [inline] Destructor.

5.20.3. Documentación de las funciones miembro

5.20.3.1. **void cruzarGenes (const [Gen](#) * *pMejor*, const [Gen](#) * *pPeor*, [Arreglo](#)< [Gen](#) > * *pHijos*, int *numHijos*, int *indice* = -1)** [virtual] Ejecuta un cruce discreto sobre genes enteros.

Parámetros:

pMejor [Gen](#) del individuo padre con mejor función de evaluación.

pPeor [Gen](#) del individuo padre con peor función de evaluación.

pHijos [Arreglo](#) de genes al cual se adicionan los genes hijos.

numHijos Numero de genes hijos a crear en el cruce.

indice Es ignorado. Existe por compatibilidad.

Implementa [OperadorCruce](#).

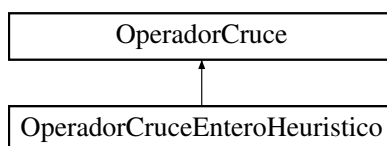
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genentero.h](#)
- [genentero.cpp](#)

5.21. Referencia de la Clase OperadorCruceEntero-Heuristico

```
#include <genentero.h>
```

Diagrama de herencias de OperadorCruceEnteroHeuristico:



5.21.1. Descripción detallada

Clase derivada de la clase [OperadorCruce](#) que define un cruce heurístico entre dos genes de tipo entero.

Crea *numHijos* genes cuyos valores corresponden a números enteros que se obtienen así:

$$h = r(pMejor - pPeor) + pMejor$$

r es un número aleatorio perteneciente al intervalo [0, 1] y *pMejor* corresponde al gen padre que tiene la mejor función de evaluación. Los valores resultantes se agregan al arreglo de genes *pHijos*

Métodos públicos

- [OperadorCruceEnteroHeuristico](#) ()

Constructor.

- [~OperadorCruceEnteroHeuristico](#) ()

Destructor.

- void `cruzarGenes` (const `Gen` *`pMejor`, const `Gen` *`pPeor`, `Arreglo`< `Gen` > *`pHijos`, int `numHijos`, int `indice`=-1)

Ejecuta un cruce heurístico sobre genes enteros.

5.21.2. Documentación del constructor y destructor

5.21.2.1. `OperadorCruceEnteroHeuristico ()` [inline] Constructor.

5.21.2.2. `~OperadorCruceEnteroHeuristico ()` [inline] Destructor.

5.21.3. Documentación de las funciones miembro

5.21.3.1. void `cruzarGenes` (const `Gen` * `pMejor`, const `Gen` * `pPeor`, `Arreglo`< `Gen` > * `pHijos`, int `numHijos`, int `indice` = -1) [virtual] Ejecuta un cruce heurístico sobre genes enteros.

Parámetros:

pMejor `Gen` del individuo padre con mejor función de evaluación.

pPeor `Gen` del individuo padre con peor función de evaluación.

pHijos `Arreglo` de genes al cual se adicionan los genes hijos.

numHijos Numero de genes hijos a crear en el cruce.

indice Es ignorado. Existe por compatibilidad.

Implementa `OperadorCruce`.

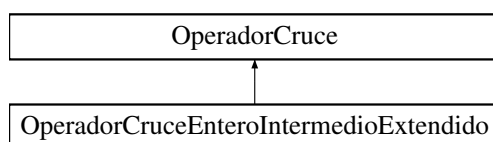
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genentero.h](#)
- [genentero.cpp](#)

5.22. Referencia de la Clase `OperadorCruceEnteroIntermedioExtendido`

```
#include <genentero.h>
```

Diagrama de herencias de `OperadorCruceEnteroIntermedioExtendido`:



5.22.1. Descripción detallada

Clase derivada de la clase `OperadorCruce` que define un cruce extendido intermedio entre dos genes de tipo entero.

Crea *numHijos* genes cuyos valores corresponden a números enteros que se obtienen así:

$$h = pMejor + \alpha_i(pPeor - pMejor)$$

α_i se escoge aleatoriamente en el intervalo $[-0,25, 1,25]$ Los valores resultantes se agregan al arreglo de genes *pHijos*

Métodos públicos

- `OperadorCruceEnteroIntermedioExtendido ()`

Constructor.

- `~OperadorCruceEnteroIntermedioExtendido ()`

Destructor.

- void `cruzarGenes` (const `Gen` *`pMejor`, const `Gen` *`pPeor`, `Arreglo`< `Gen` > *`pHijos`, int `numHijos`, int `indice`== -1)

Ejecuta un cruce intermedio extendido sobre genes enteros.

5.22.2. Documentación del constructor y destructor

5.22.2.1. `OperadorCruceEnterolIntermedioExtendido` () [inline] Constructor.

5.22.2.2. `~OperadorCruceEnterolIntermedioExtendido` () [inline] Destructor.

5.22.3. Documentación de las funciones miembro

5.22.3.1. void `cruzarGenes` (const `Gen` * `pMejor`, const `Gen` * `pPeor`, `Arreglo`< `Gen` > * `pHijos`, int `numHijos`, int `indice` = -1) [virtual] Ejecuta un cruce intermedio extendido sobre genes enteros.

Parámetros:

pMejor `Gen` del individuo padre con mejor función de evaluación.

pPeor `Gen` del individuo padre con peor función de evaluación.

pHijos `Arreglo` de genes al cual se adicionan los genes hijos.

numHijos Numero de genes hijos a crear en el cruce.

indice Es ignorado. Existe por compatibilidad.

Implementa `OperadorCruce`.

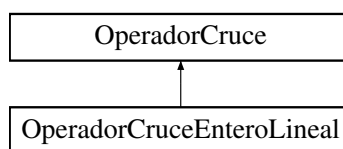
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genentero.h](#)
- [genentero.cpp](#)

5.23. Referencia de la Clase OperadorCruceEnteroLineal

```
#include <genentero.h>
```

Diagrama de herencias de OperadorCruceEnteroLineal:



5.23.1. Descripción detallada

Clase derivada de la clase [OperadorCruce](#) que define un cruce lineal entre dos genes de tipo entero.

Crea *numHijos* valores que corresponden a números enteros que se obtienen así:

$$\begin{aligned}h_1 &= \frac{1}{2}p_{Mejor} + \frac{1}{2}p_{Peor} \\h_2 &= \frac{3}{2}p_{Mejor} - \frac{1}{2}p_{Peor} \\h_3 &= -\frac{1}{2}p_{Mejor} + \frac{3}{2}p_{Peor}\end{aligned}$$

La selección de los valores que se almacenan en los genes depende del valor de *numHijos*. Los valores resultantes se agregan al arreglo de genes *pHijos*

Métodos públicos

- [OperadorCruceEnteroLineal](#) ()

Constructor.

- `~OperadorCruceEnteroLineal ()`

Destructor.

- `void cruzarGenes (const Gen *pMejor, const Gen *pPeor, Arreglo< Gen > *pHijos, int numHijos, int indice=-1)`

Ejecuta un cruce lineal sobre genes enteros.

5.23.2. Documentación del constructor y destructor

5.23.2.1. `OperadorCruceEnteroLineal ()` [inline] Constructor.

5.23.2.2. `~OperadorCruceEnteroLineal ()` [inline] Destructor.

5.23.3. Documentación de las funciones miembro

5.23.3.1. `void cruzarGenes (const Gen * pMejor, const Gen * pPeor, Arreglo< Gen > * pHijos, int numHijos, int indice = -1)` [virtual] Ejecuta un cruce lineal sobre genes enteros.

Parámetros:

pMejor Gen del individuo padre con mejor función de evaluación.

pPeor Gen del individuo padre con peor función de evaluación.

pHijos Arreglo de genes al cual se adicionan los genes hijos.

numHijos Numero de genes hijos a crear en el cruce.

indice Es ignorado. Existe por compatibilidad.

Implementa `OperadorCruce`.

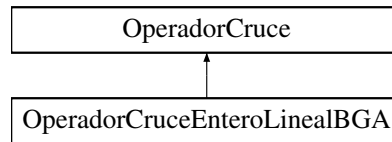
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genentero.h](#)
- [genentero.cpp](#)

5.24. Referencia de la Clase OperadorCruceEnteroLinealBGA

```
#include <genentero.h>
```

Diagrama de herencias de OperadorCruceEnteroLinealBGA:



5.24.1. Descripción detallada

Clase derivada de la clase [OperadorCruce](#) que define un cruce BGA lineal entre dos genes de tipo entero.

Crea *numHijos* genes cuyos valores corresponden a números enteros que se obtienen así:

$$h_1 = pMejor + rango_i \gamma \Delta$$

Donde:

$$\Delta = \frac{pPeor - pMejor}{|Fevaluacion_{pMejor} - Fevaluacion_{pPeor}|}$$

$$\Gamma = \sum_{k=0}^{15} \alpha_k 2^{-k}$$

$$rango_i = 0,5(Mximo - Mnimo)$$

pMejor corresponde al gen padre que tiene la mejor función de evaluación, *Máximo* y *Mínimo* corresponden a los límites establecidos en el gen y α_k puede ser

0 ó 1, con probabilidad de ser 1 de 0.0625, Los valores resultantes se agregan al arreglo de genes *pHijos*

Métodos públicos

- `OperadorCruceEnteroLinealBGA (AlgoritmoGenetico *pAG)`
Constructor.
- `~OperadorCruceEnteroLinealBGA ()`
Destructor.
- `void cruzarGenes (const Gen *pMejor, const Gen *pPeor, Arreglo< Gen > *pHijos, int numHijos, int indice=-1)`
Ejecuta el cruce lineal BGA sobre genes enteros.

Atributos privados

- `AlgoritmoGenetico * m_pAG`
Apuntador al objeto AlgoritmoGenetico en el que opera.

5.24.2. Documentación del constructor y destructor

5.24.2.1. `OperadorCruceEnteroLinealBGA (AlgoritmoGenetico * pAG)`
`[inline]` Constructor.

5.24.2.2. `~OperadorCruceEnteroLinealBGA ()` `[inline]` Destructor.

5.24.3. Documentación de las funciones miembro

5.24.3.1. `void cruzarGenes (const Gen * pMejor, const Gen * pPeor, Arreglo< Gen > * pHijos, int numHijos, int indice = -1) [virtual]` Ejecuta el cruce lineal BGA sobre genes enteros.

Parámetros:

pMejor [Gen](#) del individuo padre con mejor función de evaluación.

pPeor [Gen](#) del individuo padre con peor función de evaluación.

pHijos [Arreglo](#) de genes al cual se adicionan los genes hijos.

numHijos Numero de genes hijos a crear en el cruce.

indice Valor que identifica al individuo de cuyos genes se realiza el cruce.

Implementa [OperadorCruce](#).

5.24.4. Documentación de los datos miembro

5.24.4.1. `AlgoritmoGenetico* m_pAG [private]` Apuntador al objeto [AlgoritmoGenetico](#) en el que opera.

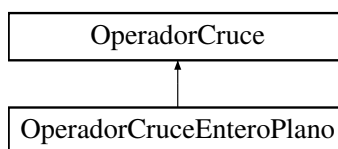
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genentero.h](#)
- [genentero.cpp](#)

5.25. Referencia de la Clase OperadorCruceEnteroPlano

```
#include <genentero.h>
```

Diagrama de herencias de OperadorCruceEnteroPlano:



5.25.1. Descripción detallada

Clase derivada de la clase [OperadorCruce](#) que define un cruce plano entre dos genes de tipo entero.

Crea *numHijos* genes cuyos valores corresponden a números enteros aleatorios que pertenecen al intervalo $[pMejor, pPeor]$ Los valores resultantes se agregan al arreglo de genes *pHijos*.

Métodos públicos

- [OperadorCruceEnteroPlano](#) ()
Constructor.
- [~OperadorCruceEnteroPlano](#) ()
Destructor.
- void [cruzarGenes](#) (const [Gen](#) *pMejor, const [Gen](#) *pPeor, [Arreglo](#)< [Gen](#) > *pHijos, int numHijos, int indice=-1)

Ejecuta un cruce plano sobre genes enteros.

5.25.2. Documentación del constructor y destructor

5.25.2.1. **OperadorCruceEnteroPlano ()** [inline] Constructor.

5.25.2.2. **~OperadorCruceEnteroPlano ()** [inline] Destructor.

5.25.3. Documentación de las funciones miembro

5.25.3.1. **void cruzarGenes (const Gen * pMejor, const Gen * pPeor, Arreglo< Gen > * pHijos, int numHijos, int indice = -1)** [virtual] Ejecuta un cruce plano sobre genes enteros.

Parámetros:

pMejor Gen del individuo padre con mejor función de evaluación.

pPeor Gen del individuo padre con peor función de evaluación.

pHijos Arreglo de genes al cual se adicionan los genes hijos.

numHijos Numero de genes hijos a crear en el cruce.

indice Es ignorado. Existe por compatibilidad.

Implementa **OperadorCruce**.

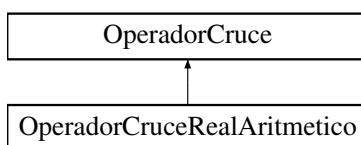
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genentero.h](#)
- [genentero.cpp](#)

5.26. Referencia de la Clase OperadorCruceRealAritmetico

```
#include <genreal.h>
```

Diagrama de herencias de OperadorCruceRealAritmetico:



5.26.1. Descripción detallada

Clase derivada de la clase [OperadorCruce](#) que define un cruce aritmético entre dos genes de tipo real.

Crea *numHijos* genes cuyos valores corresponden a números reales que se obtienen así:

$$h_1 = \lambda p_{Mejor} + (1 - \lambda) p_{Peor}$$

$$h_2 = \lambda p_{Peor} + (1 - \lambda) p_{Mejor}$$

Los valores resultantes se agregan al arreglo de genes *pHijos*

Métodos públicos

- [OperadorCruceRealAritmetico](#) (double Lambda=0.7)
- void [setLambda](#) (double Lambda)

Cambia el valor del parámetro Lambda comprobando los límites.

- double [getLambda](#) ()

Retorna el valor del parámetro Lambda.

- `~OperadorCruceRealAritmetico ()`

Destructor.

- `void cruzarGenes (const Gen *pMejor, const Gen *pPeor, Arreglo< Gen > *pHijos, int numHijos, int indice=-1)`

Ejecuta un cruce aritmético sobre genes reales.

Atributos privados

- `double m_Lambda`

Parámetro que pondera cada uno de los genes padres.

5.26.2. Documentación del constructor y destructor

5.26.2.1. `OperadorCruceRealAritmetico (double Lambda = 0.7) [inline]`

Inicializa el valor del parámetro *Lambda*.

Parámetros:

Lambda Valor que pondera cada uno de los genes padres. Por defecto es igual a 0.7.

5.26.2.2. `~OperadorCruceRealAritmetico () [inline]` Destructor.

5.26.3. Documentación de las funciones miembro

5.26.3.1. void cruzarGenes (const Gen * pMejor, const Gen * pPeor, Arreglo< Gen > * pHijos, int numHijos, int indice = -1) [virtual] Ejecuta un cruce aritmético sobre genes reales.

Parámetros:

pMejor Gen del individuo padre con mejor función de evaluación.

pPeor Gen del individuo padre con peor función de evaluación.

pHijos Arreglo de genes al cual se adicionan los genes hijos.

numHijos Numero de genes hijos a crear en el cruce.

indice Es ignorado. Existe por compatibilidad.

Implementa [OperadorCruce](#).

5.26.3.2. double getLambda () [inline] Retorna el valor del parámetro *Lambda*.

Devuelve:

Parámetro *Lambda*

5.26.3.3. void setLambda (double Lambda) [inline] Cambia el valor del parámetro *Lambda* comprobando los límites.

Parámetros:

Lambda Valor a asignar al parámetro *Lambda*.

5.26.4. Documentación de los datos miembro

5.26.4.1. double m_Lambda [private] Parámetro que pondera cada uno de los genes padres.

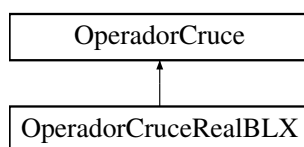
La documentación para esta clase fué generada a partir de los siguientes archivos:

- `genreal.h`
- `genreal.cpp`

5.27. Referencia de la Clase OperadorCruceRealBLX

```
#include <genreal.h>
```

Diagrama de herencias de OperadorCruceRealBLX:



5.27.1. Descripción detallada

Clase derivada de la clase [OperadorCruce](#) que define un cruce BLX - α entre dos genes de tipo real.

Crea *numHijos* genes cuyos valores corresponden a números reales aleatorios del intervalo $[c_{min} - I\alpha, c_{max} + I\alpha]$ donde $c_{min} = \min[pMejor, pPeor]$, $c_{max} = \max[pMejor, pPeor]$ e $I = c_{max} - c_{min}$. Los valores resultantes se agregan al arreglo de genes *pHijos*

Métodos públicos

- [OperadorCruceRealBLX](#) (double Alfa=0.3)

Constructor.

- void [setAlfa](#) (double Alfa)

Cambia el valor del parámetro Alfa comprobando los límites.

- double [getAlfa](#) ()

Retorna el valor del parámetro Alfa.

- `~OperadorCruceRealBLX ()`

Destructor.

- `void cruzarGenes (const Gen *pMejor, const Gen *pPeor, Arreglo< Gen > *pHijos, int numHijos, int indice=-1)`

Ejecuta el cruce BLX - α sobre genes reales.

Atributos privados

- `double m_Alfa`

Establece la amplitud del intervalo de definición.

5.27.2. Documentación del constructor y destructor

5.27.2.1. `OperadorCruceRealBLX (double Alfa = 0.3) [inline]` Constructor.

Inicializa el valor de *Alfa*

Parámetros:

Alfa Establece la amplitud del intervalo de definición. Por defecto es igual a 0.3.

5.27.2.2. `~OperadorCruceRealBLX () [inline]` Destructor.

5.27.3. Documentación de las funciones miembro

5.27.3.1. void cruzarGenes (const [Gen](#) * *pMejor*, const [Gen](#) * *pPeor*, [Arreglo](#)< [Gen](#) > * *pHijos*, int *numHijos*, int *indice* = -1) [virtual] Ejecuta el cruce BLX - α sobre genes reales.

Parámetros:

pMejor [Gen](#) del individuo padre con mejor función de evaluación.

pPeor [Gen](#) del individuo padre con peor función de evaluación.

pHijos [Arreglo](#) de genes al cual se adicionan los genes hijos.

numHijos Numero de genes hijos a crear en el cruce.

indice Es ignorado. Existe por compatibilidad.

Implementa [OperadorCruce](#).

5.27.3.2. double getAlfa () [inline] Retorna el valor del parámetro *Alfa*.

Devuelve:

Parámetro *Alfa*

5.27.3.3. void setAlfa (double *Alfa*) [inline] Cambia el valor del parámetro *Alfa* comprobando los límites.

Parámetros:

Alfa Valor a asignar al parámetro *Alfa*

5.27.4. Documentación de los datos miembro

5.27.4.1. double [m_Alfa](#) [private] Establece la amplitud del intervalo de definición.

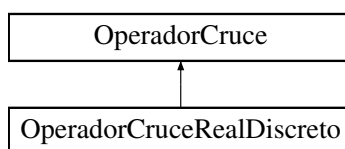
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genreal.h](#)
- [genreal.cpp](#)

5.28. Referencia de la Clase OperadorCruceRealDiscreto

```
#include <genreal.h>
```

Diagrama de herencias de OperadorCruceRealDiscreto:



5.28.1. Descripción detallada

Clase derivada de la clase [OperadorCruce](#) que define un cruce discreto entre dos genes de tipo real.

Crea *numHijos* genes cuyos valores son tomados aleatoriamente del valor del gen padre o el del gen madre. Los genes resultantes se agregan al arreglo de genes *pHijos*

Métodos públicos

- [OperadorCruceRealDiscreto](#) ()
Constructor.
- [~OperadorCruceRealDiscreto](#) ()
Destructor.
- void [cruzarGenes](#) (const [Gen](#) *pMejor, const [Gen](#) *pPeor, [Arreglo](#)< [Gen](#) > *pHijos, int numHijos, int indice=-1)

Ejecuta un cruce discreto sobre genes reales.

5.28.2. Documentación del constructor y destructor

5.28.2.1. **OperadorCruceRealDiscreto ()** [inline] Constructor.

5.28.2.2. **~OperadorCruceRealDiscreto ()** [inline] Destructor.

5.28.3. Documentación de las funciones miembro

5.28.3.1. **void cruzarGenes (const Gen * pMejor, const Gen * pPeor, Arreglo< Gen > * pHijos, int numHijos, int indice = -1)** [virtual] Ejecuta un cruce discreto sobre genes reales.

Parámetros:

pMejor Gen del individuo padre con mejor función de evaluación.

pPeor Gen del individuo padre con peor función de evaluación.

pHijos Arreglo de genes al cual se adicionan los genes hijos.

numHijos Numero de genes hijos a crear en el cruce.

indice Es ignorado. Existe por compatibilidad.

Implementa **OperadorCruce**.

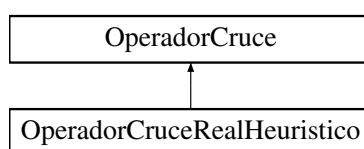
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genreal.h](#)
- [genreal.cpp](#)

5.29. Referencia de la Clase OperadorCruceReal-Heuristico

```
#include <genreal.h>
```

Diagrama de herencias de OperadorCruceRealHeuristico:



5.29.1. Descripción detallada

Clase derivada de la clase [OperadorCruce](#) que define un cruce heurístico entre dos genes de tipo real.

Crea *numHijos* genes cuyos valores corresponden a números reales que se obtienen así:

$$h = r(pMejor - pPeor) + pMejor$$

r es un número aleatorio perteneciente al intervalo [0, 1] y *pMejor* corresponde al gen padre que tiene la mejor función de evaluación. Los valores resultantes se agregan al arreglo de genes *pHijos*

Métodos públicos

- [OperadorCruceRealHeuristico](#) ()
Constructor.
- [~OperadorCruceRealHeuristico](#) ()
Destructor.

- void **cruzarGenes** (const **Gen** *pMejor, const **Gen** *pPeor, **Arreglo**< **Gen** > *pHijos, int numHijos, int indice=-1)

Ejecuta un cruce heurístico sobre genes reales.

5.29.2. Documentación del constructor y destructor

5.29.2.1. **OperadorCruceRealHeuristico ()** [inline] Constructor.

5.29.2.2. **~OperadorCruceRealHeuristico ()** [inline] Destructor.

5.29.3. Documentación de las funciones miembro

5.29.3.1. void **cruzarGenes** (const **Gen** * **pMejor**, const **Gen** * **pPeor**, **Arreglo**< **Gen** > * **pHijos**, int **numHijos**, int **indice** = -1) [virtual] Ejecuta un cruce heurístico sobre genes reales.

Parámetros:

pMejor **Gen** del individuo padre con mejor función de evaluación.

pPeor **Gen** del individuo padre con peor función de evaluación.

pHijos **Arreglo** de genes al cual se adicionan los genes hijos.

numHijos Numero de genes hijos a crear en el cruce.

indice Es ignorado. Existe por compatibilidad.

Implementa **OperadorCruce**.

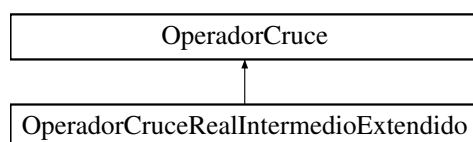
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genreal.h](#)
- [genreal.cpp](#)

5.30. Referencia de la Clase OperadorCruceRealIntermedioExtendido

```
#include <genreal.h>
```

Diagrama de herencias de OperadorCruceRealIntermedioExtendido:



5.30.1. Descripción detallada

Clase derivada de la clase [OperadorCruce](#) que define el cruce extendido intermedio entre dos genes de tipo real.

Crea *numHijos* genes cuyos valores corresponden a números reales que se obtienen así:

$$h_1 = pMejor + \alpha_i(pPeor - pMejor)$$

α_i se escoge aleatoriamente en el intervalo $[-0,25, 1,25]$ Los valores resultantes se agregan al arreglo de genes *pHijos*

Métodos públicos

- [OperadorCruceRealIntermedioExtendido](#) ()
Constructor.
- [~OperadorCruceRealIntermedioExtendido](#) ()
Destructor.

- void `cruzarGenes` (const `Gen` *`pMejor`, const `Gen` *`pPeor`, `Arreglo`< `Gen` > *`pHijos`, int `numHijos`, int `indice`== -1)

Ejecuta un cruce intermedio extendido sobre genes reales.

5.30.2. Documentación del constructor y destructor

5.30.2.1. `OperadorCruceRealIntermedioExtendido` () [inline] Constructor.

5.30.2.2. `~OperadorCruceRealIntermedioExtendido` () [inline] Destructor.

5.30.3. Documentación de las funciones miembro

5.30.3.1. void `cruzarGenes` (const `Gen` * `pMejor`, const `Gen` * `pPeor`, `Arreglo`< `Gen` > * `pHijos`, int `numHijos`, int `indice` = -1) [virtual] Ejecuta un cruce intermedio extendido sobre genes reales.

Parámetros:

pMejor `Gen` del individuo padre con mejor función de evaluación.

pPeor `Gen` del individuo padre con peor función de evaluación.

pHijos `Arreglo` de genes al cual se adicionan los genes hijos.

numHijos Numero de genes hijos a crear en el cruce.

indice Es ignorado. Existe por compatibilidad.

Implementa `OperadorCruce`.

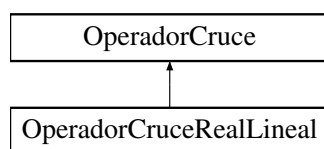
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genreal.h](#)
- [genreal.cpp](#)

5.31. Referencia de la Clase OperadorCruceRealLineal

```
#include <genreal.h>
```

Diagrama de herencias de OperadorCruceRealLineal:



5.31.1. Descripción detallada

Clase derivada de la clase [OperadorCruce](#) que define un cruce lineal entre dos genes de tipo real.

Crea *numHijos* valores que corresponden a números reales que se obtienen así:

$$h_1 = \frac{1}{2}p_{Mejor} + \frac{1}{2}p_{Peor}$$

$$h_2 = \frac{3}{2}p_{Mejor} - \frac{1}{2}p_{Peor}$$

$$h_3 = -\frac{1}{2}p_{Mejor} + \frac{3}{2}p_{Peor}$$

La selección de los valores que se almacenan en los genes depende del valor de la variable *numHijos*. Los valores resultantes se agregan al arreglo de genes *pHijos*

Métodos públicos

- [OperadorCruceRealLineal \(\)](#)

Constructor.

- `~OperadorCruceRealLineal ()`

Destructor.

- `void cruzarGenes (const Gen *madre, const Gen *padre, Arreglo< Gen > *hijos, int numHijos, int indice=-1)`

Ejecuta un cruce lineal sobre genes reales.

5.31.2. Documentación del constructor y destructor

5.31.2.1. `OperadorCruceRealLineal ()` [inline] Constructor.

5.31.2.2. `~OperadorCruceRealLineal ()` [inline] Destructor.

5.31.3. Documentación de las funciones miembro

5.31.3.1. `void cruzarGenes (const Gen * pMejor, const Gen * pPeor, Arreglo< Gen > * pHijos, int numHijos, int indice = -1)` [virtual] Ejecuta un cruce lineal sobre genes reales.

Parámetros:

pMejor Gen del individuo padre con mejor función de evaluación.

pPeor Gen del individuo padre con peor función de evaluación.

pHijos Arreglo de genes al cual se adicionan los genes hijos.

numHijos Numero de genes hijos a crear en el cruce.

indice Es ignorado. Existe por compatibilidad.

Implementa [OperadorCruce](#).

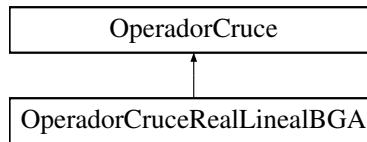
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genreal.h](#)
- [genreal.cpp](#)

5.32. Referencia de la Clase **OperadorCruceRealLinealBGA**

```
#include <genreal.h>
```

Diagrama de herencias de **OperadorCruceRealLinealBGA**:



5.32.1. Descripción detallada

Clase derivada de la clase [OperadorCruce](#) que define un cruce BGA lineal entre genes de tipo real.

Crea *numHijos* genes cuyos valores corresponden a números reales que se obtienen así:

$$h_1 = pMejor + rango_i \gamma \Delta$$

Donde:

$$\Delta = \frac{pPeor - pMejor}{|Fevaluacion_{pMejor} - Fevaluacion_{pPeor}|}$$

$$\Gamma = \sum_{k=0}^{15} \alpha_k 2^{-k}$$

$$rango_i = 0,5(Mximo - Mnimo)$$

pMejor corresponde al gen padre que tiene la mejor función de evaluación, *Máximo* y *Mínimo* corresponden a los límites establecidos en el gen y α_k puede ser

0 ó 1, con probabilidad de ser 1 de 0.0625, Los valores resultantes se agregan al arreglo de genes *pHijos*

Métodos públicos

- `OperadorCruceRealLinealBGA (AlgoritmoGenetico *pAG)`
Constructor.
- `~OperadorCruceRealLinealBGA ()`
Destructor.
- `void cruzarGenes (const Gen *madre, const Gen *padre, Arreglo< Gen > *hijos, int numHijos, int indice)`
Ejecuta el cruce BGA lineal sobre genes reales.

Atributos privados

- `AlgoritmoGenetico * m_pAG`
Apuntador al objeto AlgoritmoGenetico en el que opera.

5.32.2. Documentación del constructor y destructor

5.32.2.1. `OperadorCruceRealLinealBGA (AlgoritmoGenetico * pAG)`
`[inline]` Constructor.

5.32.2.2. `~OperadorCruceRealLinealBGA ()` `[inline]` Destructor.

5.32.3. Documentación de las funciones miembro

5.32.3.1. `void cruzarGenes (const Gen * pMejor, const Gen * pPeor, Arreglo< Gen > * pHijos, int numHijos, int indice)` [`virtual`] Ejecuta el cruce BGA lineal sobre genes reales.

Parámetros:

pMejor [Gen](#) del individuo padre con mejor función de evaluación.

pPeor [Gen](#) del individuo padre con peor función de evaluación.

pHijos [Arreglo](#) de genes al cual se adicionan los genes hijos.

numHijos Numero de genes hijos a crear en el cruce.

indice Valor que identifica al individuo de cuyos genes se realiza el cruce.

Implementa [OperadorCruce](#).

5.32.4. Documentación de los datos miembro

5.32.4.1. `AlgoritmoGenetico* m_pAG` [`private`] Apuntador al objeto [AlgoritmoGenetico](#) en el que opera.

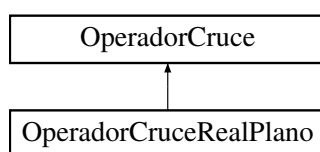
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genreal.h](#)
- [genreal.cpp](#)

5.33. Referencia de la Clase OperadorCruceRealPlano

```
#include <genreal.h>
```

Diagrama de herencias de OperadorCruceRealPlano:



5.33.1. Descripción detallada

Clase derivada de la clase [OperadorCruce](#) que define un cruce plano entre dos genes de tipo real.

Crea *numHijos* genes cuyos valores corresponden a números reales aleatorios que pertenecen al intervalo $[pMejor, pPeor]$ Los valores resultantes se agregan al arreglo de genes *pHijos*.

Métodos públicos

- [OperadorCruceRealPlano](#) ()
Constructor.
- [~OperadorCruceRealPlano](#) ()
Destructor.
- void [cruzarGenes](#) (const [Gen](#) *pMejor, const [Gen](#) *pPeor, [Arreglo](#)< [Gen](#) > *pHijos, int numHijos, int indice=-1)

Ejecuta un cruce plano sobre genes reales.

5.33.2. Documentación del constructor y destructor

5.33.2.1. **OperadorCruceRealPlano ()** [inline] Constructor.

5.33.2.2. **~OperadorCruceRealPlano ()** [inline] Destructor.

5.33.3. Documentación de las funciones miembro

5.33.3.1. **void cruzarGenes (const Gen * pMejor, const Gen * pPeor, Arreglo< Gen > * pHijos, int numHijos, int indice = -1)** [virtual] Ejecuta un cruce plano sobre genes reales.

Parámetros:

pMejor Gen del individuo padre con mejor función de evaluación.

pPeor Gen del individuo padre con peor función de evaluación.

pHijos Arreglo de genes al cual se adicionan los genes hijos.

numHijos Numero de genes hijos a crear en el cruce.

indice Es ignorado. Existe por compatibilidad.

Implementa **OperadorCruce**.

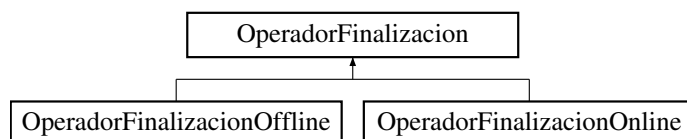
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genreal.h](#)
- [genreal.cpp](#)

5.34. Referencia de la Clase OperadorFinalizacion

```
#include <genetico.h>
```

Diagrama de herencias de OperadorFinalizacion:



5.34.1. Descripción detallada

Clase abstracta que define la estrategia de finalización del algoritmo genético.

Métodos públicos

- **OperadorFinalizacion ()**
Constructor por defecto.
- virtual **~OperadorFinalizacion ()**
Destructor.
- virtual bool **finalizar** (const **AlgoritmoGenetico** &AG)=0
Indica si el algoritmo debe finalizar.

5.34.2. Documentación del constructor y destructor

5.34.2.1. **OperadorFinalizacion ()** [inline] Constructor por defecto.

Debe sobrecargarse en las clases derivadas.

5.34.2.2. `virtual ~OperadorFinalizacion () [inline, virtual]` Destructor.

Es virtual para poder definirse en las clases derivadas.

5.34.3. Documentación de las funciones miembro

5.34.3.1. `virtual bool finalizar (const AlgoritmoGenetico & AG) [pure virtual]` Indica si el algoritmo debe finalizar.

Debe sobrecargarse en las clases derivadas

Devuelve:

true para indicar que el algoritmo debe finalizar, *false* en caso contrario

Implementado en [OperadorFinalizacionOnline](#), y [OperadorFinalizacionOffline](#).

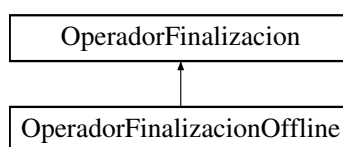
La documentación para esta clase fué generada a partir del siguiente archivo:

- [genetico.h](#)

5.35. Referencia de la Clase OperadorFinalizacionOffline

```
#include <genetico.h>
```

Diagrama de herencias de OperadorFinalizacionOffline:



5.35.1. Descripción detallada

Clase derivada de la clase [OperadorFinalizacion](#) que define la finalización del algoritmo basándose en su medida online.

Ordena la interrupción del algoritmo genético cuando éste no presenta una variación significativa en su medida offline después de un número determinado de generaciones.

Métodos públicos

- [OperadorFinalizacionOffline](#) (double FactorVariacion=0.0005, int MaxCont=30)

Constructor.

- void [setParams](#) (double FactorVariacion, int MaxCont)

Cambia el valor de los parámetros FactorVariacion y maxCont.

- void [getParams](#) (double &FactorVariacion, int &MaxCont) const

Suministra información sobre los valores de los parámetros `FactorVariacion` y `MaxCont`.

- `~OperadorFinalizacionOffline ()`

Destructor.

- `bool finalizar (const AlgoritmoGenetico &AG)`

Especifica cuándo debe darse la orden de interrupción del algoritmo genético.

Atributos privados

- `double m_factorVariacion`

Determina el factor de variación para la medida offline.

- `int m_maxCont`

Determina el número máximo de generaciones que pueden pasar sin interrumpir el algoritmo cuando la medida offline no varía en un factor mayor a `m_factorVariacion`.

- `int m_contador`

Almacena el número de generaciones que se han ejecutado sin que la medida offline varíe en un factor mayor a `m_factorVariacion`.

5.35.2. Documentación del constructor y destructor

5.35.2.1. `OperadorFinalizacionOffline (double FactorVariacion = 0.0005, int MaxCont = 30) [inline]` Constructor.

Inicializa los valores de los parámetros *FactorVariacion* y *MaxCont*

Parámetros:

FactorVariacion Determina el factor de variación para la medida offline. Por defecto es igual a 0.005.

MaxCont Determina el número máximo de generaciones que pueden pasar sin interrumpir el algoritmo cuando la medida offline no varía en un factor mayor a *FactorVariacion*. Por defecto es igual a 30.

5.35.2.2. `~OperadorFinalizacionOffline ()` [inline] Destructor.

5.35.3. Documentación de las funciones miembro

5.35.3.1. `bool finalizar (const AlgoritmoGenetico & AG)` [inline, virtual] Especifica cuándo debe darse la orden de interrupción del algoritmo genético.

Devuelve:

true si el algoritmo debe finalizar, o *false* en caso contrario.

Implementa [OperadorFinalizacion](#).

5.35.3.2. `void getParams (double & FactorVariacion, int & MaxCont) const` [inline] Suministra información sobre los valores de los parámetros *FactorVariacion* y *MaxCont*.

Parámetros:

FactorVariacion Determina el factor de variación para la medida offline.

MaxCont Determina el número máximo de generaciones que pueden pasar sin interrumpir el algoritmo cuando la medida offline no varía en un factor mayor a *FactorVariacion*.

5.35.3.3. void setParams (double *FactorVariacion*, int *MaxCont*)
[inline] Cambia el valor de los parámetros *FactorVariacion* y *maxCont*.

Parámetros:

FactorVariacion Valor a asignar al parámetro *FactorVariacion*.

MaxCont Valor a asignar al parámetro *MaxCont*.

5.35.4. Documentación de los datos miembro

5.35.4.1. int *m_contador* [private] Almacena el número de generaciones que se han ejecutado sin que la medida offline varíe en un factor mayor a *m_factorVariacion*.

5.35.4.2. double *m_factorVariacion* [private] Determina el factor de variación para la medida offline.

5.35.4.3. int *m_maxCont* [private] Determina el número máximo de generaciones que pueden pasar sin interrumpir el algoritmo cuando la medida offline no varía en un factor mayor a *m_factorVariacion*.

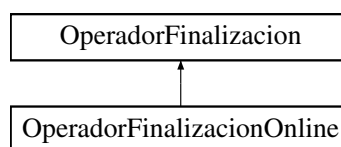
La documentación para esta clase fué generada a partir del siguiente archivo:

- [genetico.h](#)

5.36. Referencia de la Clase OperadorFinalizacionOnline

```
#include <genetico.h>
```

Diagrama de herencias de OperadorFinalizacionOnline:



5.36.1. Descripción detallada

Clase derivada de la clase [OperadorFinalizacion](#) que define la finalización del algoritmo basándose en su medida online.

Ordena la interrupción del algoritmo genético cuando éste no presenta una variación significativa en su medida online después de un número determinado de generaciones.

Métodos públicos

- [OperadorFinalizacionOnline](#) (double FactorVariacion=0.001, int MaxCont=30)

Constructor.

- void [setParams](#) (double FactorVariacion, int MaxCont)

Cambia el valor de los parámetros FactorVariacion y maxCont.

- void [getParams](#) (double &FactorVariacion, int &MaxCont) const

Suministra información sobre los valores de los parámetros `FactorVariacion` y `MaxCont`.

- `~OperadorFinalizacionOnline ()`

Destructor.

- `bool finalizar (const AlgoritmoGenetico &AG)`

Especifica cuándo debe darse la orden de interrupción del algoritmo genético.

Atributos privados

- `double m_factorVariacion`

Determina el factor de variación para la medida online.

- `int m_maxCont`

Determina el número máximo de generaciones que pueden pasar sin interrumpir el algoritmo cuando la medida online no varía en un factor mayor a `m_factorVariacion`.

- `int m_contador`

Almacena el número de generaciones que se han ejecutado sin que la medida online varíe en un factor mayor a `m_factorVariacion`.

5.36.2. Documentación del constructor y destructor

5.36.2.1. `OperadorFinalizacionOnline (double FactorVariacion = 0.001, int MaxCont = 30) [inline]` Constructor.

Inicializa los valores de los parámetros *FactorVariacion* y *MaxCont*

Parámetros:

FactorVariacion Determina el factor de variación para la medida online. Por defecto es igual a 0.001.

MaxCont Determina el número máximo de generaciones que pueden pasar sin interrumpir el algoritmo cuando la medida online no varía en un factor mayor a *FactorVariacion*. Por defecto es igual a 30.

5.36.2.2. `~OperadorFinalizacionOnline ()` `[inline]` Destructor.

5.36.3. Documentación de las funciones miembro

5.36.3.1. `bool finalizar (const AlgoritmoGenetico & AG)` `[inline, virtual]` Especifica cuándo debe darse la orden de interrupción del algoritmo genético.

Devuelve:

true si el algoritmo debe finalizar, o *false* en caso contrario.

Implementa [OperadorFinalizacion](#).

5.36.3.2. `void getParams (double & FactorVariacion, int & MaxCont)` `const [inline]` Suministra información sobre los valores de los parámetros *FactorVariacion* y *MaxCont*.

Parámetros:

FactorVariacion Determina el factor de variación para la medida online.

MaxCont Determina el número máximo de generaciones que pueden pasar sin interrumpir el algoritmo cuando la medida online no varía en un factor mayor a *FactorVariacion*.

5.36.3.3. void setParams (double *FactorVariacion*, int *MaxCont*)
[inline] Cambia el valor de los parámetros *FactorVariacion* y *maxCont*.

Parámetros:

FactorVariacion Valor a asignar al parámetro *FactorVariacion*.

MaxCont Valor a asignar al parámetro *MaxCont*.

5.36.4. Documentación de los datos miembro

5.36.4.1. int *m_contador* [private] Almacena el número de generaciones que se han ejecutado sin que la medida online varíe en un factor mayor a *m_factorVariacion*.

5.36.4.2. double *m_factorVariacion* [private] Determina el factor de variación para la medida online.

5.36.4.3. int *m_maxCont* [private] Determina el número máximo de generaciones que pueden pasar sin interrumpir el algoritmo cuando la medida online no varía en un factor mayor a *m_factorVariacion*.

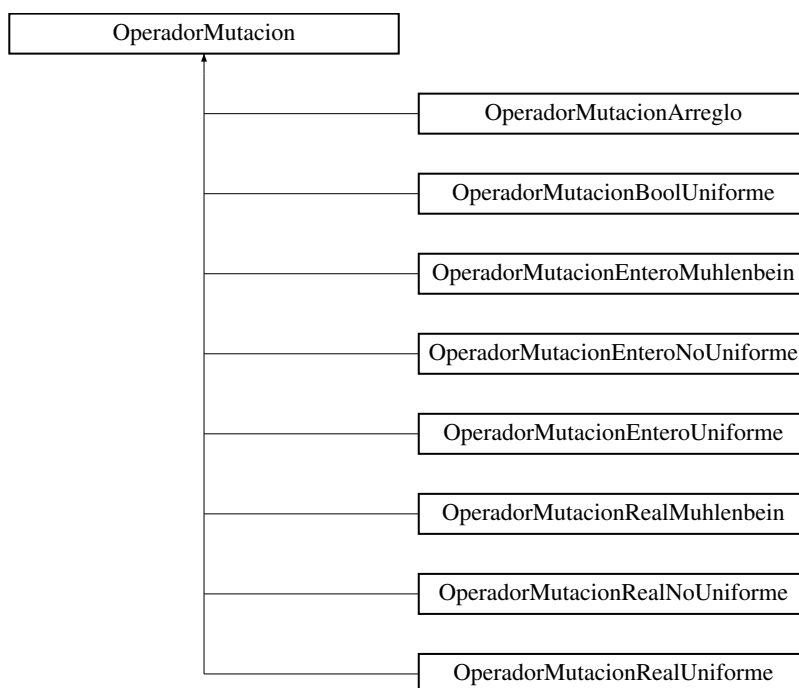
La documentación para esta clase fué generada a partir del siguiente archivo:

- [genetico.h](#)

5.37. Referencia de la Clase OperadorMutacion

```
#include <genetico.h>
```

Diagrama de herencias de OperadorMutacion:



5.37.1. Descripción detallada

Clase abstracta que administra el proceso de mutación en el algoritmo genético.

El proceso de mutación consiste en alterar la información genética de un individuo. Un objeto de una clase derivada de OperadorMutacion opera sobre un objeto de una clase derivada de [Gen](#), por lo tanto, los dos objetos deben manejar el mismo tipo de dato.

Métodos públicos

- `OperadorMutacion` (double ProbabilidadMutacion=0.1)

Constructor por defecto.

- virtual `~OperadorMutacion` ()

Destructor.

- virtual void `mutar` (Gen *pGen)

Decide si debe realizarse el proceso de mutación sobre un gen.

- double `ObtenerProbabilidadMutacion` () const

Retorna la probabilidad de mutación del gen sobre el que opera.

- virtual double `AsignarProbabilidadMutacion` (double Probabilidad)

Asigna una nueva probabilidad de mutación al gen sobre el que opera.

Atributos protegidos

- double `m_ProbabilidadMutacion`

Almacena la probabilidad de mutación del gen sobre el que opera.

Métodos privados

- virtual void `mutarGen` (Gen *pGen)=0

Efectúa el procedimiento de mutación sobre un gen.

5.37.2. Documentación del constructor y destructor

5.37.2.1. [OperadorMutacion](#) (double *ProbabilidadMutacion* = 0.1)
[inline] Constructor por defecto.

Inicializa la probabilidad de mutación para cada el objeto derivado de [Gen](#) sobre el que opera.

Parámetros:

ProbabilidadMutacion Probabilidad de mutación del gen sobre el que opera.
Por defecto es igual a 0.1

5.37.2.2. virtual ~[OperadorMutacion](#) () [inline, virtual] Destructor.

Es virtual para poder definirse en las clases derivadas.

5.37.3. Documentación de las funciones miembro

5.37.3.1. virtual double AsignarProbabilidadMutacion (double *Probabilidad*) [inline, virtual] Asigna una nueva probabilidad de mutación al gen sobre el que opera.

Parámetros:

Probabilidad Probabilidad de mutación a asignar al gen.

Devuelve:

Probabilidad de mutación asignada al gen.

Reimplementado en [OperadorMutacionArreglo](#).

5.37.3.2. virtual void mutar ([Gen](#) * *pGen*) [inline, virtual] Decide si debe realizarse el proceso de mutación sobre un gen.

La selección de los genes que deben mutar se realiza aleatoriamente teniendo en cuenta la probabilidad de mutación de cada gen.

Parámetros:

pGen Apuntador al objeto derivado de [Gen](#) sobre el que se toma la decisión.

Reimplementado en [OperadorMutacionArreglo](#).

5.37.3.3. virtual void mutarGen ([Gen](#) * *pGen*) [private, pure virtual] Efectúa el procedimiento de mutación sobre un gen.

Debe sobrecargarse en las clases derivadas

Parámetros:

pGen Apuntador al gen sobre el que opera

Implementado en [OperadorMutacionArreglo](#), [OperadorMutacionBoolUniforme](#), [OperadorMutacionEnteroUniforme](#), [OperadorMutacionEnteroNoUniforme](#), [OperadorMutacionEnteroMuhlenbein](#), [OperadorMutacionRealUniforme](#), [OperadorMutacionRealNoUniforme](#), y [OperadorMutacionRealMuhlenbein](#).

5.37.3.4. double ObtenerProbabilidadMutacion () const [inline] Retorna la probabilidad de mutación del gen sobre el que opera.

Devuelve:

Probabilidad de mutación del gen

5.37.4. Documentación de los datos miembro

5.37.4.1. double [m_ProbabilidadMutacion](#) [protected] Almacena la probabilidad de mutación del gen sobre el que opera.

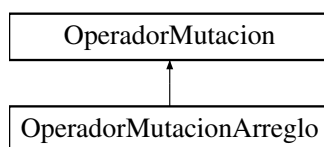
La documentación para esta clase fué generada a partir del siguiente archivo:

- [genetico.h](#)

5.38. Referencia de la Clase OperadorMutacionArreglo

```
#include <genarreglo.h>
```

Diagrama de herencias de OperadorMutacionArreglo:



5.38.1. Descripción detallada

template<class G, class T> class OperadorMutacionArreglo< G, T > Clase derivada de la clase [OperadorMutacion](#) empleada en genes de tipo arreglo. G puede ser [GenBool](#), [GenEntero](#) o [GenReal](#). T puede ser bool, long o double.

Efectúa una mutación en el gen de tipo arreglo. Cambia su tamaño dependiendo de la probabilidad de mutación. El nuevo tamaño del arreglo se genera aleatoriamente entre los límites de tamaño establecidos. Los valores de los elementos del arreglo mutan con la misma probabilidad, usando el operador de mutación por defecto para cada tipo de gen,

Métodos públicos

- [OperadorMutacionArreglo](#) (double ProbabilidadMutacion=0.01)

Constructor.

- [~OperadorMutacionArreglo](#) ()

Destructor.

- virtual double **AsignarProbabilidadMutacion** (double Probabilidad)

Asigna un nuevo valor a la probabilidad de mutación, tanto del tamaño, como de cada gen del arreglo de genes.

- void **mutar** (Gen *pGen)

Ejecuta una mutación sobre un objeto de la clase GenArreglo<G,T>.

- double **ObtenerProbabilidadMutacion** () const

Retorna la probabilidad de mutación del gen sobre el que opera.

Métodos protegidos

- void **mutarGen** (Gen *g)

*Existe por compatibilidad con la clase base. La mutación la realiza la funcion **mutar()**.*

Atributos protegidos

- **OperadorMutacion** * **m_pOperadorMutacionGenes**

Apuntador al operador de mutacion de los genes que hacen parte del arreglo.

- double **m_ProbabilidadMutacion**

Almacena la probabilidad de mutación del gen sobre el que opera.

5.38.2. Documentación del constructor y destructor

5.38.2.1. **OperadorMutacionArreglo** (double *ProbabilidadMutacion* = 0.01)

[inline] Constructor.

Inicializa el valor de la probabilidad de mutación..

Parámetros:

ProbabilidadMutacion Valor inicial para la probabilidad de mutación propia del operador. Por defecto es 0.01.

5.38.2.2. **~OperadorMutacionArreglo** () [inline] Destructor.

Destruye objetos creados

5.38.3. Documentación de las funciones miembro

5.38.3.1. **virtual double AsignarProbabilidadMutacion** (double *Probabilidad*) [inline, virtual] Asigna un nuevo valor a la probabilidad de mutación, tanto del tamaño, como de cada gen del arreglo de genes.

Parámetros:

Probabilidad Nuevo valor de probabilidad de mutación. Se restringe al rango [0,1]

Devuelve:

Valor de probabilidad de mutacion asignado.

Reimplementado de [OperadorMutacion](#).

5.38.3.2. **void mutar** ([Gen](#) * *pGen*) [virtual] Ejecuta una mutación sobre un objeto de la clase GenArreglo<G,T>.

Método sobrecargado de la clase operador mutacion.

Parámetros:

pGen Apuntador al objeto de la clase GenArreglo<G,T> que será sometido a mutación.

Reimplementado de [OperadorMutacion](#).

5.38.3.3. void mutarGen (Gen * g) [inline, protected, virtual]
Existe por compatibilidad con la clase base. La mutación la realiza la funcion [mutar\(\)](#).

Implementa [OperadorMutacion](#).

5.38.3.4. double ObtenerProbabilidadMutacion () const [inline, inherited] Retorna la probabilidad de mutación del gen sobre el que opera.

Devuelve:

Probabilidad de mutación del gen

5.38.4. Documentación de los datos miembro

5.38.4.1. [OperadorMutacion*](#) [m_pOperadorMutacionGenes](#) [protected] Apuntador al operador de mutacion de los genes que hacen parte del arreglo.

5.38.4.2. double [m_ProbabilidadMutacion](#) [protected, inherited]
Almacena la probabilidad de mutación del gen sobre el que opera.

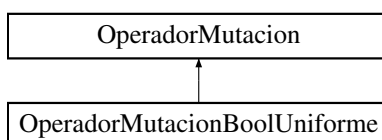
La documentación para esta clase fué generada a partir del siguiente archivo:

- [genarreglo.h](#)

5.39. Referencia de la Clase OperadorMutacion-BoolUniforme

```
#include <genbool.h>
```

Diagrama de herencias de OperadorMutacionBoolUniforme:



5.39.1. Descripción detallada

Clase derivada de la clase [OperadorMutacion](#) que efectúa una mutación uniforme sobre un gen de tipo booleano.

El gen cambia su valor de *false* a *true* o viceversa

Métodos públicos

- [OperadorMutacionBoolUniforme](#) (double ProbabilidadMutacion=0.1)

Constructor.

- [~OperadorMutacionBoolUniforme](#) ()

Destructor.

- void [mutarGen](#) ([Gen](#) *pGen)

Ejecuta una mutación uniforme sobre un objeto de la clase [GenBool](#).

- virtual void [mutar](#) ([Gen](#) *pGen)

Decide si debe realizarse el proceso de mutación sobre un gen.

- double **ObtenerProbabilidadMutacion** () const

Retorna la probabilidad de mutación del gen sobre el que opera.

- virtual double **AsignarProbabilidadMutacion** (double Probabilidad)

Asigna una nueva probabilidad de mutación al gen sobre el que opera.

Atributos protegidos

- double **m_ProbabilidadMutacion**

Almacena la probabilidad de mutación del gen sobre el que opera.

5.39.2. Documentación del constructor y destructor

5.39.2.1. **OperadorMutacionBoolUniforme (double *ProbabilidadMutacion* = 0.1) [inline]** Constructor.

Inicializa el valor de la probabilidad de mutación.

Parámetros:

ProbabilidadMutacion Establece el valor inicial para la probabilidad de mutación propia del operador. Por defecto es 0.1.

5.39.2.2. **~OperadorMutacionBoolUniforme () [inline]** Destructor.

5.39.3. Documentación de las funciones miembro

5.39.3.1. virtual double AsignarProbabilidadMutacion (double *Probabilidad*) [*inline, virtual, inherited*] Asigna una nueva probabilidad de mutación al gen sobre el que opera.

Parámetros:

Probabilidad Probabilidad de mutación a asignar al gen.

Devuelve:

Probabilidad de mutación asignada al gen.

Reimplementado en [OperadorMutacionArreglo](#).

5.39.3.2. virtual void mutar ([Gen](#) * *pGen*) [*inline, virtual, inherited*] Decide si debe realizarse el proceso de mutación sobre un gen.

La selección de los genes que deben mutar se realiza aleatoriamente teniendo en cuenta la probabilidad de mutación de cada gen.

Parámetros:

pGen Apuntador al objeto derivado de [Gen](#) sobre el que se toma la decisión.

Reimplementado en [OperadorMutacionArreglo](#).

5.39.3.3. void mutarGen ([Gen](#) * *pGen*) [*virtual*] Ejecuta una mutación uniforme sobre un objeto de la clase [GenBool](#).

Parámetros:

pGen Apuntador al objeto de la clase [GenBool](#) que será sometido a mutación.

Implementa [OperadorMutacion](#).

5.39.3.4. double ObtenerProbabilidadMutacion () const [inline, inherited] Retorna la probabilidad de mutación del gen sobre el que opera.

Devuelve:

Probabilidad de mutación del gen

5.39.4. Documentación de los datos miembro

5.39.4.1. double m_ProbabilidadMutacion [protected, inherited] Almacena la probabilidad de mutación del gen sobre el que opera.

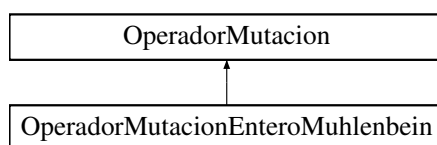
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genbool.h](#)
- [genbool.cpp](#)

5.40. Referencia de la Clase OperadorMutacion-EnteroMuhlenbein

```
#include <genentero.h>
```

Diagrama de herencias de OperadorMutacionEnteroMuhlenbein:



5.40.1. Descripción detallada

Clase derivada de la clase [OperadorMutacion](#) que define una mutación Muhlenbein sobre un gen de de tipo entero.

Al aplicar la mutación de Muhlenbein a un gen c_i que debe pertenecer al intervalo $[a_i, b_i]$, el nuevo valor del gen es:

$$c'_i = c_i \pm \gamma rango$$

$$\gamma = \sum_{i=0}^{15} \alpha_i 2^{-i}$$

donde *rango* define el rango de mutación y se emplea como $Factor(b_i - a_i)$ El signo + ó - se escoge aleatoriamente con igual probabilidad, y α_i puede ser 0 ó 1, con probabilidad de ser 1 igual a: $P(\alpha_i = 1) = \frac{1}{16}$

Métodos públicos

- [OperadorMutacionEnteroMuhlenbein](#) (double ProbabilidadMutacion=0.1, double Factor=0.1)

Constructor.

- void **setFactor** (double Factor)

Cambia el valor del parámetro Factor.

- double **getFactor** () const

Retorna el valor del parámetro Factor.

- **~OperadorMutacionEnteroMuhlenbein** ()

Destructor.

- void **mutarGen** (Gen *pGen)

*Ejecuta una mutación Muhlenbein sobre un objeto de la clase **GenEntero**.*

- virtual void **mutar** (Gen *pGen)

Decide si debe realizarse el proceso de mutación sobre un gen.

- double **ObtenerProbabilidadMutacion** () const

Retorna la probabilidad de mutación del gen sobre el que opera.

- virtual double **AsignarProbabilidadMutacion** (double Probabilidad)

Asigna una nueva probabilidad de mutación al gen sobre el que opera.

Atributos protegidos

- double **m_ProbabilidadMutacion**

Almacena la probabilidad de mutación del gen sobre el que opera.

Atributos privados

- double `m_Factor`

Valor que define el rango de mutación. Solo puede modificarse en el constructor.

5.40.2. Documentación del constructor y destructor

5.40.2.1. `OperadorMutacionEnteroMuhlenbein` (double *ProbabilidadMutacion* = 0.1, double *Factor* = 0.1) [inline] Constructor.

Inicializa el valor de la probabilidad de mutación y el valor del factor que define el rango de mutación.

Parámetros:

ProbabilidadMutacion Valor inicial para la probabilidad de mutación propia del operador. Por defecto es 0.1.

Factor Valor que define el rango de mutación. Por defecto es 0.1.

5.40.2.2. `~OperadorMutacionEnteroMuhlenbein` () [inline] Destructor.

5.40.3. Documentación de las funciones miembro

5.40.3.1. `virtual double AsignarProbabilidadMutacion` (double *Probabilidad*) [inline, virtual, inherited] Asigna una nueva probabilidad de mutación al gen sobre el que opera.

Parámetros:

Probabilidad Probabilidad de mutación a asignar al gen.

Devuelve:

Probabilidad de mutación asignada al gen.

Reimplementado en [OperadorMutacionArreglo](#).

5.40.3.2. double getFactor () const [inline] Retorna el valor del parámetro Factor.

5.40.3.3. virtual void mutar (Gen * pGen) [inline, virtual, inherited] Decide si debe realizarse el proceso de mutación sobre un gen.

La selección de los genes que deben mutar se realiza aleatoriamente teniendo en cuenta la probabilidad de mutación de cada gen.

Parámetros:

pGen Apuntador al objeto derivado de [Gen](#) sobre el que se toma la decisión.

Reimplementado en [OperadorMutacionArreglo](#).

5.40.3.4. void mutarGen (Gen * pGen) [virtual] Ejecuta una mutación Muhlenbein sobre un objeto de la clase [GenEntero](#).

Parámetros:

pGen Apuntador al objeto de la clase [GenEntero](#) que será sometido a mutación.

Implementa [OperadorMutacion](#).

5.40.3.5. double ObtenerProbabilidadMutacion () const [inline, inherited] Retorna la probabilidad de mutación del gen sobre el que opera.

Devuelve:

Probabilidad de mutación del gen

5.40.3.6. void setFactor (double *Factor*) [`inline`] Cambia el valor del parámetro *Factor*.

Parámetros:

Factor Valor que define el rango de mutación.

5.40.4. Documentación de los datos miembro

5.40.4.1. double `m_Factor` [`private`] Valor que define el rango de mutación. Solo puede modificarse en el constructor.

5.40.4.2. double `m_ProbabilidadMutacion` [`protected, inherited`] Almacena la probabilidad de mutación del gen sobre el que opera.

La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genentero.h](#)
- [genentero.cpp](#)

5.41. Referencia de la Clase OperadorMutacionEnteroNoUniforme

```
#include <genentero.h>
```

Diagrama de herencias de OperadorMutacionEnteroNoUniforme:



5.41.1. Descripción detallada

Clase derivada de la clase [OperadorMutacion](#) que define una mutación no uniforme sobre un gen de tipo entero.

Al aplicar la mutación no uniforme a un gen c_i que debe pertenecer al intervalo $[a_i, b_i]$ en la generación t , con un número máximo de generaciones g_{max} , el nuevo valor del gen es:

$$c'_i = \begin{cases} c_i + \Delta(t, b_i - c_i) & \text{si } \tau > 0 \\ c_i - \Delta(t, b_i - c_i) & \text{si } \tau \leq 0 \end{cases}$$

Donde

$$\Delta(t, y) = y \left(1 - r^{\left(1 - \frac{t}{g_{max}}\right)^b} \right)$$

b es un parametro seleccionable por el usuario, por defecto es 0.5; r es un número aleatorio en el intervalo $[0, 1]$, τ es un número aleatorio que puede valer 0 ó 1.

Métodos públicos

- `OperadorMutacionEnteroNoUniforme` (`AlgoritmoGenetico *pAG`, `double ProbabilidadMutacion=0.1`, `double b=0.5`)

Constructor.

- `void setB` (`double b`)

Cambia el valor del parámetro b.

- `double getB` () `const`

Retorna el valor del parámetro b.

- `~OperadorMutacionEnteroNoUniforme` ()

Destructor.

- `void mutarGen` (`Gen *pGen`)

Ejecuta una mutación no uniforme sobre un objeto de la clase `GenEntero`.

- `virtual void mutar` (`Gen *pGen`)

Decide si debe realizarse el proceso de mutación sobre un gen.

- `double ObtenerProbabilidadMutacion` () `const`

Retorna la probabilidad de mutación del gen sobre el que opera.

- `virtual double AsignarProbabilidadMutacion` (`double Probabilidad`)

Asigna una nueva probabilidad de mutación al gen sobre el que opera.

Atributos protegidos

- double `m_ProbabilidadMutacion`

Almacena la probabilidad de mutación del gen sobre el que opera.

Atributos privados

- double `m_b`

**Parámetro que determina el grado de dependencia con el numero de generaciones.*

- `AlgoritmoGenetico * m_pAG`

Apuntador a un objeto de la clase `AlgoritmoGenetico`.

5.41.2. Documentación del constructor y destructor

5.41.2.1. `OperadorMutacionEnteroNoUniforme` (`AlgoritmoGenetico * pAG`, double *ProbabilidadMutacion* = 0.1, double *b* = 0.5) [inline] Constructor.

Inicializa el valor de la probabilidad de mutación y el valor que determina el grado de dependencia con el numero de generaciones.

Parámetros:

pAG Apuntador al algoritmo genético sobre el que opera

ProbabilidadMutacion Valor inicial para la probabilidad de mutación propia del operador. Por defecto es 0.1

b Valor Valor que determina el grado de dependencia con el numero de generaciones. Por defecto es 0.5

5.41.2.2. `~OperadorMutacionEnteroNoUniforme () [inline]` Destructor.

5.41.3. Documentación de las funciones miembro

5.41.3.1. `virtual double AsignarProbabilidadMutacion (double Probabilidad) [inline, virtual, inherited]` Asigna una nueva probabilidad de mutación al gen sobre el que opera.

Parámetros:

Probabilidad Probabilidad de mutación a asignar al gen.

Devuelve:

Probabilidad de mutación asignada al gen.

Reimplementado en [OperadorMutacionArreglo](#).

5.41.3.2. `double getB () const [inline]` Retorna el valor del parámetro *b*.

5.41.3.3. `virtual void mutar (Gen * pGen) [inline, virtual, inherited]` Decide si debe realizarse el proceso de mutación sobre un gen.

La selección de los genes que deben mutar se realiza aleatoriamente teniendo en cuenta la probabilidad de mutación de cada gen.

Parámetros:

pGen Apuntador al objeto derivado de [Gen](#) sobre el que se toma la decisión.

Reimplementado en [OperadorMutacionArreglo](#).

5.41.3.4. `void mutarGen (Gen * pGen) [virtual]` Ejecuta una mutación no uniforme sobre un objeto de la clase [GenEntero](#).

Parámetros:

pGen Apuntador al objeto de la clase [GenEntero](#) que será sometido a mutación.

Implementa [OperadorMutacion](#).

5.41.3.5. double ObtenerProbabilidadMutacion () const [inline, inherited] Retorna la probabilidad de mutación del gen sobre el que opera.

Devuelve:

Probabilidad de mutación del gen

5.41.3.6. void setB (double b) [inline] Cambia el valor del parámetro *b*.

Parámetros:

b Valor a asignar al parámetro *b*.

5.41.4. Documentación de los datos miembro

5.41.4.1. double m_b [private] *Parámetro que determina el grado de dependencia con el numero de generaciones.

5.41.4.2. AlgoritmoGenetico* m_pAG [private] Apuntador a un objeto de la clase [AlgoritmoGenetico](#).

5.41.4.3. double m_ProbabilidadMutacion [protected, inherited] Almacena la probabilidad de mutación del gen sobre el que opera.

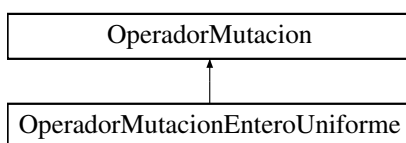
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genentero.h](#)
- [genentero.cpp](#)

5.42. Referencia de la Clase OperadorMutacionEnteroUniforme

```
#include <genentero.h>
```

Diagrama de herencias de OperadorMutacionEnteroUniforme:



5.42.1. Descripción detallada

Clase derivada de la clase [OperadorMutacion](#) que define una mutación uniforme sobre un gen de tipo entero.

El nuevo valor del gen es un número entero aleatorio restringido al rango definido por los límites establecidos para el gen.

Métodos públicos

- [OperadorMutacionEnteroUniforme](#) (double ProbabilidadMutacion=0.1)

Constructor.

- [~OperadorMutacionEnteroUniforme](#) ()

Destructor.

- void [mutarGen](#) ([Gen](#) *pGen)

Ejecuta una mutación uniforme sobre un objeto de la clase [GenEntero](#).

- virtual void **mutar** (**Gen** *pGen)

Decide si debe realizarse el proceso de mutación sobre un gen.

- double **ObtenerProbabilidadMutacion** () const

Retorna la probabilidad de mutación del gen sobre el que opera.

- virtual double **AsignarProbabilidadMutacion** (double Probabilidad)

Asigna una nueva probabilidad de mutación al gen sobre el que opera.

Atributos protegidos

- double **m_ProbabilidadMutacion**

Almacena la probabilidad de mutación del gen sobre el que opera.

5.42.2. Documentación del constructor y destructor

5.42.2.1. **OperadorMutacionEnteroUniforme (double **ProbabilidadMutacion = 0.1**) [inline]** Constructor.

Inicializa el valor de la probabilidad de mutación.

Parámetros:

ProbabilidadMutacion Establece el valor inicial para la probabilidad de mutación propia del operador. Por defecto es 0.1.

5.42.2.2. **~OperadorMutacionEnteroUniforme () [inline]** Destructor.

5.42.3. Documentación de las funciones miembro

5.42.3.1. virtual double AsignarProbabilidadMutacion (double *Probabilidad*) [*inline, virtual, inherited*] Asigna una nueva probabilidad de mutación al gen sobre el que opera.

Parámetros:

Probabilidad Probabilidad de mutación a asignar al gen.

Devuelve:

Probabilidad de mutación asignada al gen.

Reimplementado en [OperadorMutacionArreglo](#).

5.42.3.2. virtual void mutar ([Gen](#) * *pGen*) [*inline, virtual, inherited*] Decide si debe realizarse el proceso de mutación sobre un gen.

La selección de los genes que deben mutar se realiza aleatoriamente teniendo en cuenta la probabilidad de mutación de cada gen.

Parámetros:

pGen Apuntador al objeto derivado de [Gen](#) sobre el que se toma la decisión.

Reimplementado en [OperadorMutacionArreglo](#).

5.42.3.3. void mutarGen ([Gen](#) * *pGen*) [*virtual*] Ejecuta una mutación uniforme sobre un objeto de la clase [GenEntero](#).

Parámetros:

pGen Apuntador al objeto de la clase [GenEntero](#) que será sometido a mutación.

Implementa [OperadorMutacion](#).

5.42.3.4. double ObtenerProbabilidadMutacion () const [inline, inherited] Retorna la probabilidad de mutación del gen sobre el que opera.

Devuelve:

Probabilidad de mutación del gen

5.42.4. Documentación de los datos miembro

5.42.4.1. double m_ProbabilidadMutacion [protected, inherited] Almacena la probabilidad de mutación del gen sobre el que opera.

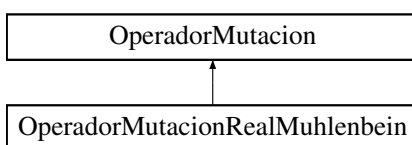
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genentero.h](#)
- [genentero.cpp](#)

5.43. Referencia de la Clase OperadorMutacion-RealMuhlenbein

```
#include <genreal.h>
```

Diagrama de herencias de OperadorMutacionRealMuhlenbein:



5.43.1. Descripción detallada

Clase derivada de la clase [OperadorMutacion](#) que define una mutación Muhlenbein sobre un gen de de tipo real.

Al aplicar la mutación de Muhlenbein a un gen c_i que debe pertenecer al intervalo $[a_i, b_i]$, el nuevo valor del gen es:

$$c'_i = c_i \pm \gamma rango$$

$$\gamma = \sum_{i=0}^{15} \alpha_i 2^{-i}$$

donde *rango* define el rango de mutación y se emplea como $Factor(b_i - a_i)$ El signo + ó - se escoge aleatoriamente con igual probabilidad, y α_i puede ser 0 ó 1, con probabilidad de ser 1 igual a: $P(\alpha_i = 1) = \frac{1}{16}$

Métodos públicos

- [OperadorMutacionRealMuhlenbein](#) (double ProbabilidadMutacion=0.1, double Factor=0.1)

Constructor.

- void **setFactor** (double Factor)

Cambia el valor del parámetro Factor.

- double **getFactor** ()

Retorna el valor del parámetro Factor.

- **~OperadorMutacionRealMuhlenbein** ()

Destructor.

- void **mutarGen** (Gen *pGen)

*Ejecuta una mutación Muhlenbein sobre un objeto de la clase **GenReal**.*

- virtual void **mutar** (Gen *pGen)

Decide si debe realizarse el proceso de mutación sobre un gen.

- double **ObtenerProbabilidadMutacion** () const

Retorna la probabilidad de mutación del gen sobre el que opera.

- virtual double **AsignarProbabilidadMutacion** (double Probabilidad)

Asigna una nueva probabilidad de mutación al gen sobre el que opera.

Atributos protegidos

- double **m_ProbabilidadMutacion**

Almacena la probabilidad de mutación del gen sobre el que opera.

Atributos privados

- double `m_Factor`

Valor que define el rango de mutación.

5.43.2. Documentación del constructor y destructor

5.43.2.1. `OperadorMutacionRealMuhlenbein` (double *ProbabilidadMutacion* = 0.1, double *Factor* = 0.1) [inline] Constructor.

Inicializa el valor de la probabilidad de mutación y el valor del factor que define el rango de mutación.

Parámetros:

ProbabilidadMutacion Valor inicial para la probabilidad de mutación propia del operador. Por defecto es 0.1.

Factor Valor que define el rango de mutación. Por defecto es 0.1.

5.43.2.2. `~OperadorMutacionRealMuhlenbein` () [inline] Destructor.

5.43.3. Documentación de las funciones miembro

5.43.3.1. `virtual double AsignarProbabilidadMutacion` (double *Probabilidad*) [inline, virtual, inherited] Asigna una nueva probabilidad de mutación al gen sobre el que opera.

Parámetros:

Probabilidad Probabilidad de mutación a asignar al gen.

Devuelve:

Probabilidad de mutación asignada al gen.

Reimplementado en [OperadorMutacionArreglo](#).

5.43.3.2. double getFactor () [*inline*] Retorna el valor del parámetro *Factor*.

Devuelve:

parámetro *Factor*

5.43.3.3. virtual void mutar (Gen * pGen) [*inline, virtual, inherited*] Decide si debe realizarse el proceso de mutación sobre un gen.

La selección de los genes que deben mutar se realiza aleatoriamente teniendo en cuenta la probabilidad de mutación de cada gen.

Parámetros:

pGen Apuntador al objeto derivado de [Gen](#) sobre el que se toma la decisión.

Reimplementado en [OperadorMutacionArreglo](#).

5.43.3.4. void mutarGen (Gen * pGen) [*virtual*] Ejecuta una mutación Muhlenbein sobre un objeto de la clase [GenReal](#).

Parámetros:

pGen Apuntador al objeto de la clase [GenReal](#) que será sometido a mutación.

Implementa [OperadorMutacion](#).

5.43.3.5. double ObtenerProbabilidadMutacion () **const** [*inline, inherited*] Retorna la probabilidad de mutación del gen sobre el que opera.

Devuelve:

Probabilidad de mutación del gen

5.43.3.6. void setFactor (double *Factor*) [inline] Cambia el valor del parámetro *Factor*.

Parámetros:

Factor Valor que define el rango de mutación.

5.43.4. Documentación de los datos miembro

5.43.4.1. double [m_Factor](#) [private] Valor que define el rango de mutación.

5.43.4.2. double [m_ProbabilidadMutacion](#) [protected, inherited]
Almacena la probabilidad de mutación del gen sobre el que opera.

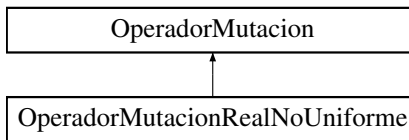
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genreal.h](#)
- [genreal.cpp](#)

5.44. Referencia de la Clase OperadorMutacion-RealNoUniforme

```
#include <genreal.h>
```

Diagrama de herencias de OperadorMutacionRealNoUniforme:



5.44.1. Descripción detallada

Clase derivada de la clase [OperadorMutacion](#) que define una mutación no uniforme sobre un gen de tipo real.

Al aplicar la mutación no uniforme a un gen c_i que debe pertenecer al intervalo $[a_i, b_i]$ en la generación t , con un número máximo de generaciones g_{max} , el nuevo valor del gen es:

$$c'_i = \begin{cases} c_i + \Delta(t, b_i - c_i) & \text{si } \tau > 0 \\ c_i - \Delta(t, b_i - c_i) & \text{si } \tau \leq 0 \end{cases}$$

Donde

$$\Delta(t, y) = y \left(1 - r^{\left(1 - \frac{t}{g_{max}}\right)^b} \right)$$

b es un parametro seleccionable por el usuario, por defecto es 0.5; r es un número aleatorio en el intervalo $[0, 1]$, τ es un número aleatorio que puede valer 0 ó 1.

Métodos públicos

- `OperadorMutacionRealNoUniforme` (`AlgoritmoGenetico *pAG`, `double ProbabilidadMutacion=0.1`, `double b=0.5`)

Constructor.

- `void setB` (`double b`)

Cambia el valor del parámetro b.

- `double getB` ()

Retorna el valor del parámetro b.

- `~OperadorMutacionRealNoUniforme` ()

Destructor.

- `void mutarGen` (`Gen *g`)

Ejecuta una mutación no uniforme sobre un objeto de la clase `GenReal`.

- `virtual void mutar` (`Gen *pGen`)

Decide si debe realizarse el proceso de mutación sobre un gen.

- `double ObtenerProbabilidadMutacion` () `const`

Retorna la probabilidad de mutación del gen sobre el que opera.

- `virtual double AsignarProbabilidadMutacion` (`double Probabilidad`)

Asigna una nueva probabilidad de mutación al gen sobre el que opera.

Atributos protegidos

- double `m_ProbabilidadMutacion`

Almacena la probabilidad de mutación del gen sobre el que opera.

Atributos privados

- double `m_b`

Parámetro que determina el grado de dependencia con el numero de generaciones.

- `AlgoritmoGenetico * m_pAG`

Apuntador a un objeto de la clase `AlgoritmoGenetico`.

5.44.2. Documentación del constructor y destructor

5.44.2.1. `OperadorMutacionRealNoUniforme (AlgoritmoGenetico * pAG, double ProbabilidadMutacion = 0.1, double b = 0.5) [inline]` Constructor.

Inicializa el valor de la probabilidad de mutación y el valor *b* de que determina el grado de dependencia con el numero de generaciones.

Parámetros:

pAG Apuntador al algoritmo genético sobre el que opera

ProbabilidadMutacion Valor inicial para la probabilidad de mutación propia del operador. Por defecto es 0.1

b Valor que determina el grado de dependencia con el numero de generaciones. Por defecto es 0.5

5.44.2.2. `~OperadorMutacionRealNoUniforme ()` `[inline]` Destructor.

5.44.3. Documentación de las funciones miembro

5.44.3.1. `virtual double AsignarProbabilidadMutacion (double Probabilidad)` `[inline, virtual, inherited]` Asigna una nueva probabilidad de mutación al gen sobre el que opera.

Parámetros:

Probabilidad Probabilidad de mutación a asignar al gen.

Devuelve:

Probabilidad de mutación asignada al gen.

Reimplementado en [OperadorMutacionArreglo](#).

5.44.3.2. `double getB ()` `[inline]` Retorna el valor del parámetro *b*.

return Parámetro *b*.

5.44.3.3. `virtual void mutar (Gen * pGen)` `[inline, virtual, inherited]` Decide si debe realizarse el proceso de mutación sobre un gen.

La selección de los genes que deben mutar se realiza aleatoriamente teniendo en cuenta la probabilidad de mutación de cada gen.

Parámetros:

pGen Apuntador al objeto derivado de [Gen](#) sobre el que se toma la decisión.

Reimplementado en [OperadorMutacionArreglo](#).

5.44.3.4. void mutarGen (Gen * pGen) [virtual] Ejecuta una mutación no uniforme sobre un objeto de la clase [GenReal](#).

Parámetros:

pGen Apuntador al objeto de la clase [GenReal](#) que será sometido a mutación.

Implementa [OperadorMutacion](#).

5.44.3.5. double ObtenerProbabilidadMutacion () const [inline, inherited] Retorna la probabilidad de mutación del gen sobre el que opera.

Devuelve:

Probabilidad de mutación del gen

5.44.3.6. void setB (double b) [inline] Cambia el valor del parámetro *b*.

Parámetros:

b Valor a asignar al parámetro *b*.

5.44.4. Documentación de los datos miembro

5.44.4.1. double m_b [private] Parámetro que determina el grado de dependencia con el numero de generaciones.

5.44.4.2. [AlgoritmoGenetico*](#) m_pAG [private] Apuntador a un objeto de la clase [AlgoritmoGenetico](#).

5.44.4.3. double [m_ProbabilidadMutacion](#) [protected, inherited]

Almacena la probabilidad de mutación del gen sobre el que opera.

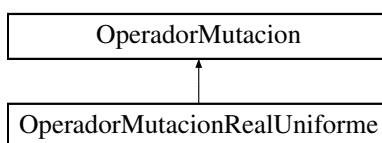
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genreal.h](#)
- [genreal.cpp](#)

5.45. Referencia de la Clase OperadorMutacion-RealUniforme

```
#include <genreal.h>
```

Diagrama de herencias de OperadorMutacionRealUniforme:



5.45.1. Descripción detallada

Clase derivada de la clase [OperadorMutacion](#) que define una mutación uniforme sobre un gen de tipo real.

El nuevo valor del gen es un número real aleatorio restringido al rango definido por los límites establecidos en el gen.

Métodos públicos

- [OperadorMutacionRealUniforme](#) (double ProbabilidadMutacion=0.1)

Constructor.

- [~OperadorMutacionRealUniforme](#) ()

Destructor.

- void [mutarGen](#) ([Gen](#) *pGen)

Ejecuta una mutación uniforme sobre un objeto de la clase [GenReal](#).

- virtual void **mutar** (**Gen** *pGen)

Decide si debe realizarse el proceso de mutación sobre un gen.

- double **ObtenerProbabilidadMutacion** () const

Retorna la probabilidad de mutación del gen sobre el que opera.

- virtual double **AsignarProbabilidadMutacion** (double Probabilidad)

Asigna una nueva probabilidad de mutación al gen sobre el que opera.

Atributos protegidos

- double **m_ProbabilidadMutacion**

Almacena la probabilidad de mutación del gen sobre el que opera.

5.45.2. Documentación del constructor y destructor

5.45.2.1. **OperadorMutacionRealUniforme (double *ProbabilidadMutacion* = 0.1) [inline]** Constructor.

Inicializa el valor de la probabilidad de mutación.

Parámetros:

ProbabilidadMutacion Establece el valor inicial para la probabilidad de mutación propia del operador. Por defecto es 0.1.

5.45.2.2. **~OperadorMutacionRealUniforme () [inline]** Destructor.

5.45.3. Documentación de las funciones miembro

5.45.3.1. virtual double AsignarProbabilidadMutacion (double *Probabilidad*) [*inline, virtual, inherited*] Asigna una nueva probabilidad de mutación al gen sobre el que opera.

Parámetros:

Probabilidad Probabilidad de mutación a asignar al gen.

Devuelve:

Probabilidad de mutación asignada al gen.

Reimplementado en [OperadorMutacionArreglo](#).

5.45.3.2. virtual void mutar ([Gen](#) * *pGen*) [*inline, virtual, inherited*] Decide si debe realizarse el proceso de mutación sobre un gen.

La selección de los genes que deben mutar se realiza aleatoriamente teniendo en cuenta la probabilidad de mutación de cada gen.

Parámetros:

pGen Apuntador al objeto derivado de [Gen](#) sobre el que se toma la decisión.

Reimplementado en [OperadorMutacionArreglo](#).

5.45.3.3. void mutarGen ([Gen](#) * *pGen*) [*virtual*] Ejecuta una mutación uniforme sobre un objeto de la clase [GenReal](#).

Parámetros:

pGen Apuntador al objeto de la clase [GenReal](#) que será sometido a mutación.

Implementa [OperadorMutacion](#).

5.45.3.4. double ObtenerProbabilidadMutacion () const [inline, inherited] Retorna la probabilidad de mutación del gen sobre el que opera.

Devuelve:

Probabilidad de mutación del gen

5.45.4. Documentación de los datos miembro

5.45.4.1. double m_ProbabilidadMutacion [protected, inherited] Almacena la probabilidad de mutación del gen sobre el que opera.

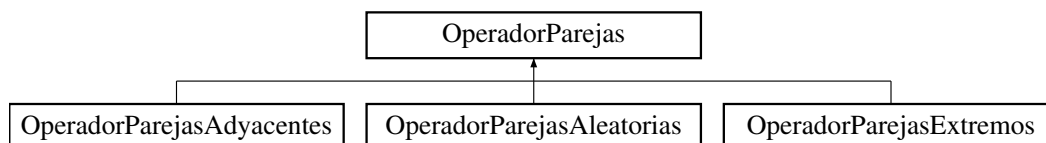
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genreal.h](#)
- [genreal.cpp](#)

5.46. Referencia de la Clase OperadorParejas

```
#include <genetico.h>
```

Diagrama de herencias de OperadorParejas:



5.46.1. Descripción detallada

Clase abstracta que administra la asignación de parejas para cada individuo de la población.

Para cada individuo presente en la población se asigna otro individuo que servirá de pareja para intercambiar su información genética.

Métodos públicos

- **OperadorParejas ()**

Constructor por defecto.

- **virtual ~OperadorParejas ()**

Destructor.

- **virtual void asignarParejas (Poblacion &Pob)=0**

Asigna las parejas a los individuos de la población.

5.46.2. Documentación del constructor y destructor

5.46.2.1. **OperadorParejas ()** `[inline]` Constructor por defecto.

Debe sobrecargarse en las clases derivadas

5.46.2.2. **virtual ~OperadorParejas ()** `[inline, virtual]` Destructor.

Es virtual para poder definirse en las clases derivadas

5.46.3. Documentación de las funciones miembro

5.46.3.1. **virtual void asignarParejas (Poblacion & Pob)** `[pure virtual]`

Asigna las parejas a los individuos de la población.

Debe sobrecargarse en las clases derivadas.

Parámetros:

Pob Referencia a la población sobre la que opera.

Implementado en [OperadorParejasAleatorias](#), [OperadorParejasAdyacentes](#), y [OperadorParejasExtremos](#).

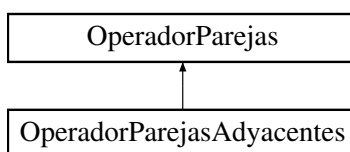
La documentación para esta clase fué generada a partir del siguiente archivo:

- [genetico.h](#)

5.47. Referencia de la Clase OperadorParejas-Adyacentes

```
#include <genetico.h>
```

Diagrama de herencias de OperadorParejasAdyacentes:



5.47.1. Descripción detallada

Clase derivada de la clase [OperadorParejas](#) que define la asignación de parejas adyacentes para los individuos de la población.

A cada individuo de la población se le asigna otro individuo que corresponderá a su pareja en el proceso de reproducción. Al individuo i de la población se le asigna como pareja el individuo $i+1$, el proceso continúa con los individuos a los que aún no se les ha asignado pareja. Cuando el numero de individuos es impar, al último individuo de la población no se le asigna una pareja.

Métodos públicos

- [OperadorParejasAdyacentes](#) ()

Constructor.

- [~OperadorParejasAdyacentes](#) ()

Destructor.

- void [asignarParejas](#) ([Poblacion](#) &Pob)

Asigna parejas adyacentes para los individuos de la población.

5.47.2. Documentación del constructor y destructor

5.47.2.1. [OperadorParejasAdyacentes](#) () [inline] Constructor.

5.47.2.2. ~[OperadorParejasAdyacentes](#) () [inline] Destructor.

5.47.3. Documentación de las funciones miembro

5.47.3.1. void [asignarParejas](#) ([Poblacion](#) & *Pob*) [virtual] Asigna parejas adyacentes para los individuos de la población.

Parámetros:

Pob Referencia a la población sobre la que opera.

Implementa [OperadorParejas](#).

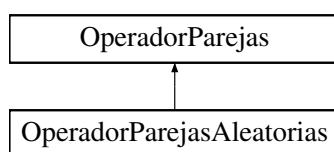
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genetico.h](#)
- [genetico.cpp](#)

5.48. Referencia de la Clase OperadorParejasAleatorias

```
#include <genetico.h>
```

Diagrama de herencias de OperadorParejasAleatorias:



5.48.1. Descripción detallada

Clase derivada de [OperadorParejas](#) que define la asignación de parejas aleatorias para los individuos de la población.

A cada individuo de la población se le asigna otro individuo que corresponderá a su pareja en el proceso de reproducción. La asignación se realiza aleatoriamente teniendo en cuenta que no deben asignarse individuos que ya tengan pareja establecida y que un individuo no puede ser asignado como su propia pareja. Cuando el numero de individuos es impar, a un individuo de la población no se le asigna pareja.

Métodos públicos

- [OperadorParejasAleatorias \(\)](#)

Constructor.

- [~OperadorParejasAleatorias \(\)](#)

Destructor.

- void [asignarParejas](#) ([Poblacion](#) &Pob)

Asigna parejas aleatoriamente para los individuos de la población.

5.48.2. Documentación del constructor y destructor

5.48.2.1. [OperadorParejasAleatorias](#) () [inline] Constructor.

5.48.2.2. [~OperadorParejasAleatorias](#) () [inline] Destructor.

5.48.3. Documentación de las funciones miembro

5.48.3.1. void [asignarParejas](#) ([Poblacion](#) & *Pob*) [virtual] Asigna parejas aleatoriamente para los individuos de la población.

Parámetros:

Pob Referencia a la población sobre la que opera.

Implementa [OperadorParejas](#).

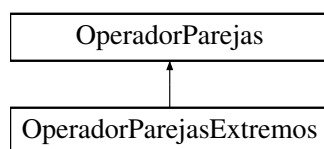
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genetico.h](#)
- [genetico.cpp](#)

5.49. Referencia de la Clase OperadorParejas-Extremos

```
#include <genetico.h>
```

Diagrama de herencias de OperadorParejasExtremos:



5.49.1. Descripción detallada

Clase derivada de la clase [OperadorParejas](#) que define la asignación de parejas extremas para los individuos de la población.

A cada individuo de la población se le asigna otro individuo que corresponderá a su pareja en el proceso de reproducción. Al primer individuo del arreglo de la población se le asigna como pareja el último individuo del arreglo, al segundo individuo se le asigna el penúltimo. El proceso continúa sucesivamente hasta asignar parejas a todos los individuos de la generación. Cuando el numero de individuos es impar, al individuo ubicado en el centro de la población no se le asigna pareja.

Métodos públicos

- [OperadorParejasExtremos \(\)](#)

Constructor.

- [~OperadorParejasExtremos \(\)](#)

Destructor.

- void [asignarParejas](#) ([Poblacion](#) &Pob)

Asigna parejas extremas para los individuos de la población.

5.49.2. Documentación del constructor y destructor

5.49.2.1. [OperadorParejasExtremos](#) () [inline] Constructor.

5.49.2.2. [~OperadorParejasExtremos](#) () [inline] Destructor.

5.49.3. Documentación de las funciones miembro

5.49.3.1. void [asignarParejas](#) ([Poblacion](#) & *Pob*) [virtual] Asigna parejas extremas para los individuos de la población.

Parámetros:

Pob Referencia a la población sobre la que opera.

Implementa [OperadorParejas](#).

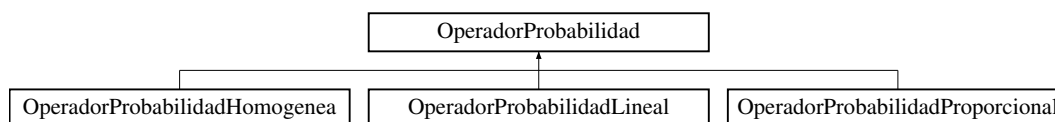
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genetico.h](#)
- [genetico.cpp](#)

5.50. Referencia de la Clase OperadorProbabilidad

```
#include <genetico.h>
```

Diagrama de herencias de OperadorProbabilidad:



5.50.1. Descripción detallada

Clase abstracta que administra la asignación de la probabilidad de supervivencia a los individuos de la población.

Generalmente, la probabilidad de supervivencia se asigna con base en la función de evaluación de cada individuo.

Métodos públicos

- **OperadorProbabilidad ()**

Constructor por defecto.

- **virtual ~OperadorProbabilidad ()**

Destructor.

- **virtual void asignarProbabilidad (Poblacion &Pob, bool Maximizar)=0**

Asigna una probabilidad de supervivencia a los individuos de la población.

5.50.2. Documentación del constructor y destructor

5.50.2.1. [OperadorProbabilidad](#) () `[inline]` Constructor por defecto.

Debe sobrecargarse en las clases derivadas

5.50.2.2. `virtual ~`[OperadorProbabilidad](#) () `[inline, virtual]` Destructor.

Es virtual para poder definirse en las clases derivadas

5.50.3. Documentación de las funciones miembro

5.50.3.1. `virtual void` [asignarProbabilidad](#) ([Poblacion](#) & *Pob*, bool *Maximizar*) `[pure virtual]` Asigna una probabilidad de supervivencia a los individuos de la población.

Debe sobrecargarse en las clases derivadas

Parámetros:

Pob Referencia a la población sobre la que opera.

Maximizar Si su valor es *true* a los individuos se les asignan probabilidades de supervivencia directamente proporcionales a su función objetivo, de lo contrario se asignan probabilidades inversamente proporcionales.

Implementado en [OperadorProbabilidadProporcional](#), [OperadorProbabilidadLineal](#), y [OperadorProbabilidadHomogenea](#).

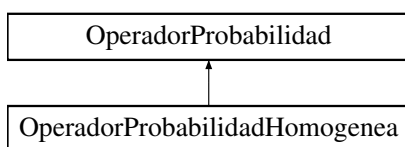
La documentación para esta clase fue generada a partir del siguiente archivo:

- [genetico.h](#)

5.51. Referencia de la Clase Operador-ProbabilidadHomogenea

```
#include <genetico.h>
```

Diagrama de herencias de OperadorProbabilidadHomogenea:



5.51.1. Descripción detallada

Clase derivada de la clase [OperadorProbabilidad](#) que define el proceso de asignación de probabilidad de supervivencia homogénea.

En una población de N individuos, se asigna la misma probabilidad de supervivencia a todos ellos, dada por:

$$P(i) = \frac{1}{N}$$

Métodos públicos

- [OperadorProbabilidadHomogenea](#) ()

Constructor.

- [~OperadorProbabilidadHomogenea](#) ()

Destructor.

- void [asignarProbabilidad](#) ([Poblacion](#) &Pob, bool Maximizar)

Efectúa el proceso de asignación de probabilidad de supervivencia homogénea.

5.51.2. Documentación del constructor y destructor

5.51.2.1. [OperadorProbabilidadHomogenea \(\)](#) [inline] Constructor.

5.51.2.2. [~OperadorProbabilidadHomogenea \(\)](#) [inline] Destructor.

5.51.3. Documentación de las funciones miembro

5.51.3.1. **void asignarProbabilidad ([Poblacion](#) & *Pob*, bool *Maximizar*)**
[virtual] Efectúa el proceso de asignación de probabilidad de supervivencia homogénea.

Parámetros:

Pob Referencia a la población sobre la que opera

Maximizar No es utilizado. Existe por compatibilidad

Implementa [OperadorProbabilidad](#).

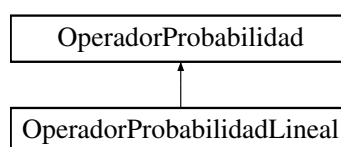
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genetico.h](#)
- [genetico.cpp](#)

5.52. Referencia de la Clase Operador-ProbabilidadLineal

```
#include <genetico.h>
```

Diagrama de herencias de OperadorProbabilidadLineal:



5.52.1. Descripción detallada

Clase derivada de la clase [OperadorProbabilidad](#) que define el proceso de asignación de probabilidad de supervivencia lineal.

En una población de N individuos, se ordenan de mejor a peor y se asigna a cada uno una probabilidad de supervivencia que depende de su posición en la población, así:

$$P(i) = \frac{\eta_{max} - \frac{(\eta_{max} - \eta_{min})i}{N+1}}{N}$$

Donde,

$$\eta_{max} = 2 - \eta_{min}$$

η_{min} se escoge aleatoriamente del intervalo $[0, 2]$.

Métodos públicos

- [OperadorProbabilidadLineal](#) (double Nmin=0.5)

Constructor.

- void **setNmin** (double Nmin)

Cambia el valor del parámetro Nmin comprobando los límites.

- double **getNmin** () const

Retorna el valor del parámetro Nmin.

- **~OperadorProbabilidadLineal** ()

Destructor.

- void **asignarProbabilidad** (Poblacion &Pob, bool Maximizar)

Efectúa el proceso de asignación de probabilidad de supervivencia lineal.

Atributos protegidos

- double **m_nmin**

5.52.2. Documentación del constructor y destructor

5.52.2.1. **OperadorProbabilidadLineal** (double *Nmin* = 0.5) [inline]

Constructor.

Parámetros:

Nmin Valor que pondera el grado de proporcionalidad. Por defecto es igual a 0.5.

5.52.2.2. **~OperadorProbabilidadLineal** () [inline] Destructor.

5.52.3. Documentación de las funciones miembro

5.52.3.1. void asignarProbabilidad ([Poblacion](#) & *Pob*, bool *Maximizar*)
[virtual] Efectúa el proceso de asignación de probabilidad de supervivencia lineal.

Parámetros:

Pob Referencia a la población sobre la que opera

Maximizar Su valor sirve como referencia para determinar qué individuo es mejor a otro dependiendo de su función de evaluación.

Implementa [OperadorProbabilidad](#).

5.52.3.2. double getNmin () const [inline] Retorna el valor del parámetro *Nmin*.

Devuelve:

Valor del parámetro *Nmin*

5.52.3.3. void setNmin (double *Nmin*) [inline] Cambia el valor del parámetro *Nmin* comprobando los límites.

Parámetros:

Nmin Valor a asignar al parámetro *Nmin*

5.52.4. Documentación de los datos miembro

5.52.4.1. double [m_nmin](#) [protected] Valor que pondera el grado de proporcionalidad

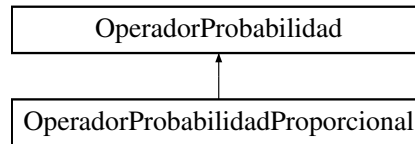
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genetico.h](#)
- [genetico.cpp](#)

5.53. Referencia de la Clase Operador-ProbabilidadProporcional

```
#include <genetico.h>
```

Diagrama de herencias de OperadorProbabilidadProporcional:



5.53.1. Descripción detallada

Clase derivada de la clase [OperadorProbabilidad](#) que define el proceso de asignación de probabilidad de supervivencia proporcional a cada individuo de la población.

En una población de N individuos, se asigna una probabilidad de supervivencia a cada individuo proporcional al valor de su función objetivo $FObj(i)$, así:

- Al minimizar:

$$P(i) = \frac{\frac{1}{FObjProp(i)}}{\sum_{i=1}^N \frac{1}{FObj(i)}}$$

- Al maximizar:

$$P(i) = \frac{FObjProp(i)}{\sum_{i=1}^N FObj(i)}$$

Donde,

$$FObjProp(i) = FObj(i) + (1 - FObjmin)$$

$$FObjmin = \min \{1, FObj(i)\}, \quad i = 1, \dots, N.$$

Métodos públicos

- `OperadorProbabilidadProporcional ()`

Constructor.

- `~OperadorProbabilidadProporcional ()`

Destructor.

- `void asignarProbabilidad (Poblacion &Pob, bool Maximizar)`

Efectúa el proceso de asignación de probabilidad de supervivencia proporcional a cada individuo de la población.

5.53.2. Documentación del constructor y destructor

5.53.2.1. `OperadorProbabilidadProporcional ()` [inline] Constructor.

5.53.2.2. `~OperadorProbabilidadProporcional ()` [inline] Destructor.

5.53.3. Documentación de las funciones miembro

5.53.3.1. `void asignarProbabilidad (Poblacion & Pob, bool Maximizar)`

[virtual] Efectúa el proceso de asignación de probabilidad de supervivencia proporcional a cada individuo de la población.

Parámetros:

Pob Referencia a la población sobre la que opera

Maximizar Su valor sirve como referencia para determinar qué individuo es mejor a otro dependiendo de su función de evaluación.

Implementa [OperadorProbabilidad](#).

La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genetico.h](#)
- [genetico.cpp](#)

5.54. Referencia de la Clase Operador-Reproduccion

```
#include <genetico.h>
```

Diagrama de herencias de OperadorReproduccion:



5.54.1. Descripción detallada

Clase abstracta que define la estrategia general de reproducción del algoritmo.

La estrategia general de reproducción define cuáles individuos conformarán la siguiente generación del algoritmo.

Métodos públicos

- `OperadorReproduccion ()`

Constructor por defecto.

- `virtual ~OperadorReproduccion ()`

Destructor.

- `virtual void reproducir (Poblacion &Pob, bool Maximizar)=0`

Ejecuta la estrategia general reproducción para una población.

5.54.2. Documentación del constructor y destructor

5.54.2.1. **OperadorReproduccion ()** [inline] Constructor por defecto.

Debe sobrecargarse en las clases derivadas.

5.54.2.2. **virtual ~OperadorReproduccion ()** [inline, virtual] Destructor.

Es virtual para poder definirse en las clases derivadas.

5.54.3. Documentación de las funciones miembro

5.54.3.1. **virtual void reproducir (Poblacion & Pob, bool Maximizar)** [pure virtual] Ejecuta la estrategia general reproducción para una población.

Debe sobrecargarse en las clases derivadas.

Parámetros:

Pob Referencia a la población sobre la que opera.

Maximizar Su valor sirve como referencia para determinar qué individuo es mejor a otro dependiendo de su función de evaluación.

Implementado en [OperadorReproduccionCruceSimple](#), [OperadorReproduccionDosPadresDosHijos](#), [OperadorReproduccionMejorPadreMejorHijo](#), y [OperadorReproduccionMejoresEntrePadresEHijos](#).

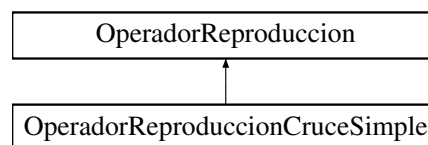
La documentación para esta clase fué generada a partir del siguiente archivo:

- [genetico.h](#)

5.55. Referencia de la Clase OperadorReproduccionCruceSimple

```
#include <genetico.h>
```

Diagrama de herencias de OperadorReproduccionCruceSimple:



5.55.1. Descripción detallada

Clase derivada de la clase [OperadorReproduccion](#) que define el cruce simple entre dos individuos de la población.

El cruce simple genera dos individuos hijos por cada cruce entre dos individuos padres intercambiando los genes respectivos en un punto de cruce. El punto de cruce se selecciona aleatoriamente para cada pareja en el intervalo $[1, n-1]$ donde n corresponde al número de genes en el genoma. A partir del gen siguiente al punto de cruce hasta el final del genoma se intercambian los genes del individuo padre con los de la madre, estos nuevos individuos corresponden a los dos hijos generados que reemplazarán a sus padres en la siguiente generación del algoritmo.

Métodos públicos

- [OperadorReproduccionCruceSimple \(\)](#)

Constructor.

- `~OperadorReproduccionCruceSimple ()`

Destructor.

- `void reproducir (Poblacion &Pob, bool Maximizar)`

Ejecuta el cruce simple entre dos individuos de la población.

5.55.2. Documentación del constructor y destructor

5.55.2.1. `OperadorReproduccionCruceSimple ()` `[inline]` Constructor.

5.55.2.2. `~OperadorReproduccionCruceSimple ()` `[inline]` Destructor.

5.55.3. Documentación de las funciones miembro

5.55.3.1. `void reproducir (Poblacion & Pob, bool Maximizar)` `[virtual]`

Ejecuta el cruce simple entre dos individuos de la población.

Parámetros:

Pob Referencia a la población sobre la que opera

Maximizar No se utiliza. Existe por compatibilidad

Implementa `OperadorReproduccion`.

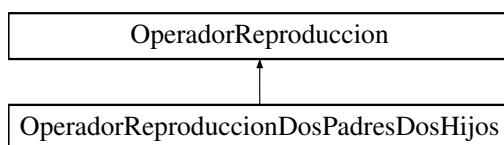
La documentación para esta clase fué generada a partir de los siguientes archivos:

- `genetico.h`
- `genetico.cpp`

5.56. Referencia de la Clase Operador-ReproduccionDosPadresDosHijos

```
#include <genetico.h>
```

Diagrama de herencias de OperadorReproduccionDosPadresDosHijos:



5.56.1. Descripción detallada

Clase derivada de la clase [OperadorReproduccion](#) que define la estrategia de reproducción dos padres dos hijos.

Por cada pareja de individuos que efectúen el cruce se crean dos hijos que reemplazarán a sus padres en la siguiente iteración del algoritmo

Métodos públicos

- [OperadorReproduccionDosPadresDosHijos](#) ()

Constructor.

- [~OperadorReproduccionDosPadresDosHijos](#) ()

Destructor de la clase OperadorReproduccionDosPadresDosHijos.

- void [reproducir](#) ([Poblacion](#) &Pob, bool Maximizar)

Ejecuta la estrategia de reproducción dos padres dos hijos.

5.56.2. Documentación del constructor y destructor

5.56.2.1. [OperadorReproduccionDosPadresDosHijos](#) () [inline] Constructor.

5.56.2.2. [~OperadorReproduccionDosPadresDosHijos](#) () [inline] Destructor de la clase [OperadorReproduccionDosPadresDosHijos](#).

5.56.3. Documentación de las funciones miembro

5.56.3.1. void reproducir ([Poblacion](#) & *Pob*, bool *Maximizar*) [virtual] Ejecuta la estrategia de reproducción dos padres dos hijos.

Parámetros:

Pob Referencia a la población sobre la que opera

Maximizar Su valor sirve como referencia para determinar qué individuo es mejor a otro dependiendo de su función de evaluación.

Implementa [OperadorReproduccion](#).

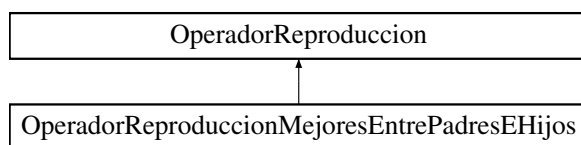
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genetico.h](#)
- [genetico.cpp](#)

5.57. Referencia de la Clase OperadorReproduccionMejoresEntrePadresEHijos

```
#include <genetico.h>
```

Diagrama de herencias de OperadorReproduccionMejoresEntrePadresEHijos:



5.57.1. Descripción detallada

Clase derivada de la clase [OperadorReproduccion](#) que define la estrategia de reproducción mejores entre padres e hijos.

Por cada pareja de individuos que efectúen el cruce se crean dos hijos. Los individuos que conformarán la siguiente generación serán los dos que tengan la mejor función de evaluación entre los cuatro.

Métodos públicos

- [OperadorReproduccionMejoresEntrePadresEHijos](#) ()

Constructor.

- [~OperadorReproduccionMejoresEntrePadresEHijos](#) ()

Destructor.

- void [reproducir](#) ([Poblacion](#) &Pob, bool Maximizar)

Ejecuta la estrategia de reproducción mejores entre padres e hijos.

5.57.2. Documentación del constructor y destructor

5.57.2.1. **OperadorReproduccionMejoresEntrePadresEHijos** () [inline]
Constructor.

5.57.2.2. **~OperadorReproduccionMejoresEntrePadresEHijos** ()
[inline] Destructor.

5.57.3. Documentación de las funciones miembro

5.57.3.1. **void reproducir (Poblacion & Pob, bool Maximizar)** [virtual]
Ejecuta la estrategia de reproducción mejores entre padres e hijos.

Parámetros:

Pob Referencia a la población sobre la que opera

Maximizar Su valor sirve como referencia para determinar qué individuo es mejor a otro dependiendo de su función de evaluación.

Implementa [OperadorReproduccion](#).

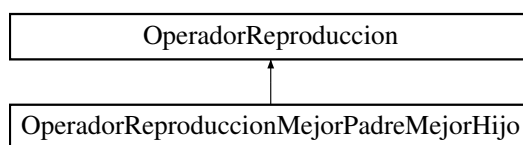
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genetico.h](#)
- [genetico.cpp](#)

5.58. Referencia de la Clase Operador-ReproduccionMejorPadreMejorHijo

```
#include <genetico.h>
```

Diagrama de herencias de OperadorReproduccionMejorPadreMejorHijo:



5.58.1. Descripción detallada

Clase derivada de la clase [OperadorReproduccion](#) que define la estrategia de reproducción mejor padre mejor hijo.

Por cada pareja de individuos que efectúen el cruce se crean dos hijos. El mejor individuo hijo reemplazará al peor individuo padre en la siguiente iteración del algoritmo

Métodos públicos

- [OperadorReproduccionMejorPadreMejorHijo](#) ()

Constructor.

- [~OperadorReproduccionMejorPadreMejorHijo](#) ()

Destructor.

- void [reproducir](#) ([Poblacion](#) &Pob, bool Maximizar)

Ejecuta la estrategia de reproducción mejor padre mejor hijo.

5.58.2. Documentación del constructor y destructor

5.58.2.1. [OperadorReproduccionMejorPadreMejorHijo](#) `() [inline]`
Constructor.

5.58.2.2. `~`[OperadorReproduccionMejorPadreMejorHijo](#) `() [inline]`
Destructor.

5.58.3. Documentación de las funciones miembro

5.58.3.1. `void reproducir` ([Poblacion](#) & *Pob*, bool *Maximizar*) `[virtual]`
Ejecuta la estrategia de reproducción mejor padre mejor hijo.

Parámetros:

Pob Referencia a la población sobre la que opera

Maximizar Su valor sirve como referencia para determinar qué individuo es mejor a otro dependiendo de su función de evaluación.

Implementa [OperadorReproduccion](#).

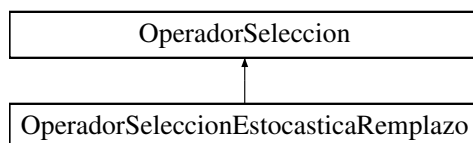
La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genetico.h](#)
- [genetico.cpp](#)

5.59. Referencia de la Clase OperadorSeleccion

```
#include <genetico.h>
```

Diagrama de herencias de OperadorSeleccion:



5.59.1. Descripción detallada

Clase abstracta que administra el proceso de selección de individuos en la población.

Un objeto de una clase derivada de OperadorSeleccion selecciona un conjunto de individuos presentes en la generación actual que servirán de base para formar la nueva generación del algoritmo.

Métodos públicos

- **OperadorSeleccion** ()

Constructor por defecto.

- virtual **~OperadorSeleccion** ()

Destructor.

- virtual void **seleccionar** (**Poblacion** &pPob)=0

Efectúa el proceso de selección en la población.

5.59.2. Documentación del constructor y destructor

5.59.2.1. **OperadorSeleccion ()** [inline] Constructor por defecto.

Debe sobrecargarse en las clases derivadas

5.59.2.2. **virtual ~OperadorSeleccion ()** [inline, virtual] Destructor.

Es virtual para poder definirse en las clases derivadas

5.59.3. Documentación de las funciones miembro

5.59.3.1. **virtual void seleccionar (Poblacion & pPob)** [pure virtual]

Efectúa el proceso de selección en la población.

Debe sobrecargarse en las clases derivadas

Parámetros:

pPob Referencia a la población sobre la que opera.

Implementado en [OperadorSeleccionEstocasticaRemplazo](#).

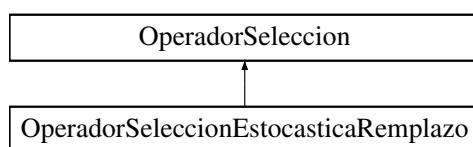
La documentación para esta clase fué generada a partir del siguiente archivo:

- [genetico.h](#)

5.60. Referencia de la Clase OperadorSeleccionEstocasticaReemplazo

```
#include <genetico.h>
```

Diagrama de herencias de OperadorSeleccionEstocasticaReemplazo:



5.60.1. Descripción detallada

Clase derivada de la clase [OperadorSeleccion](#) que define el proceso de selección estocástica con reemplazo de los individuos.

Los individuos que componen la población son ordenados como segmentos de una línea, cada segmento corresponde a la probabilidad de supervivencia del individuo. La selección se produce al generarse un valor aleatorio que coincida con el segmento del individuo. El proceso se repite hasta obtener el número de individuos deseados, creando así la base para la nueva población.

Métodos públicos

- [OperadorSeleccionEstocasticaReemplazo \(\)](#)
Constructor.
- [~OperadorSeleccionEstocasticaReemplazo \(\)](#)
Destructor.

- void [seleccionar](#) ([Poblacion](#) &pPob)

Ejecuta el proceso de selección estocástica con reemplazo de los individuos de la plobación.

5.60.2. Documentación del constructor y destructor

5.60.2.1. [OperadorSeleccionEstocasticaReemplazo](#) () [inline] Constructor.

5.60.2.2. [~OperadorSeleccionEstocasticaReemplazo](#) () [inline] Destructor.

5.60.3. Documentación de las funciones miembro

5.60.3.1. void [seleccionar](#) ([Poblacion](#) & *pPob*) [virtual] Ejecuta el proceso de selección estocástica con reemplazo de los individuos de la plobación.

Parámetros:

pPob Referencia a la población sobre la que opera.

Implementa [OperadorSeleccion](#).

La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genetico.h](#)
- [genetico.cpp](#)

5.61. Referencia de la Clase Poblacion

```
#include <genetico.h>
```

5.61.1. Descripción detallada

Clase que administra la información de los individuos de un algoritmo genético.

Un objeto de la clase Poblacion contiene un arreglo de apuntadores a objetos de la clase [Individuo](#). Además contiene los métodos necesarios para realizar las operaciones sobre la población en cada iteración del algoritmo genético

Métodos públicos

- [Poblacion](#) ([AlgoritmoGenetico](#) *pAG, int nIndividuos, bool crearAleatorios=true)

Constructor por defecto.

- [Poblacion](#) (const [Poblacion](#) &origen)

Constructor por copia de otra poblacion.

- virtual ~[Poblacion](#) ()

Destructor.

- const [Poblacion](#) & operator= (const [Poblacion](#) &origen)

Operador de asignación a partir de otra población.

- [AlgoritmoGenetico](#) * [GetAG](#) () const

Retorna un apuntador al objeto de la clase [AlgoritmoGenetico](#) al que pertenece la población.

- void **ordenar** (bool Maximizar=false)
Ordena los miembros de la población según su función de evaluación.
- void **SetAG** (**AlgoritmoGenetico** *Ag)
Cambia el algoritmo genetico al que pertenece la población.
- **Individuo** & **getIndividuo** (int pos) const
Devuelve el individuo ubicado en una posición determinada de la población.
- int **InsertarIndividuo** (**Individuo** *ind, int indice=-1)
Inserta un individuo en una posición determinada de la población.
- **Individuo** * **reemplazarIndividuo** (**Individuo** *pNuevoInd, int pos)
Reemplaza el individuo ubicado en una posición determinada de la población por otro individuo.
- int **getTam** () const
Retorna el número de individuos presentes en la población.
- int **setTam** (int nuevoTam, bool reemplazar=false, bool crear-Aleatorios=false)
Cambia el tamaño de la población.
- void **mutar** ()
Ordena a la población que pase por el proceso de mutación.

Atributos protegidos

- `AlgoritmoGenetico * m_pAG`

Apuntador al objeto de la clase `AlgoritmoGenetico` al que pertenece la población.

- `Arreglo< Individuo > * m_pGeneracion`

Arreglo de objetos de la clase `Individuo` que conforman la población.

5.61.2. Documentación del constructor y destructor

5.61.2.1. `Poblacion (AlgoritmoGenetico * pAG, int nIndividuos, bool crearAleatorios = true)` Constructor por defecto.

5.61.2.2. `Poblacion (const Poblacion & origen)` Constructor por copia de otra población.

Construye el nuevo objeto copiando idénticamente las propiedades de otro objeto de la clase `Poblacion`.

Parámetros:

origen Objeto del que se hace copia.

5.61.2.3. `~Poblacion ()` `[virtual]` Destructor.

Elimina todos los objetos creados dinámicamente

5.61.3. Documentación de las funciones miembro

5.61.3.1. `AlgoritmoGenetico* GetAG () const` `[inline]` Retorna un apuntador al objeto de la clase `AlgoritmoGenetico` al que pertenece la población.

Devuelve:

Apuntador al objeto de la clase [AlgoritmoGenetico](#) al que pertenece la población

5.61.3.2. [Individuo](#)& getIndividuo (int *pos*) const [inline] Devuelve el individuo ubicado en una posición determinada de la población.

Parámetros:

pos Posición del individuo a obtener

Devuelve:

referencia al individuo ubicado en la posición *pos*.

5.61.3.3. int getTam () const [inline] Retorna el número de individuos presentes en la población.

return Tamaño de la población

5.61.3.4. int InsertarIndividuo ([Individuo](#) * *ind*, int *pos* = -1) Inserta un individuo en una posición determinada de la población.

Parámetros:

ind Apuntador al individuo a insertar

pos Posición en la que será insertado el individuo. Si no se especifica, el individuo se ubicará en la última posición.

Devuelve:

Posición en la que fue ubicado el individuo.

5.61.3.5. void mutar () Ordena a la población que pase por el proceso de mutación.

Hace que cada uno de los individuos presentes en población realice el proceso de mutación.

5.61.3.6. const Poblacion & operator= (const Poblacion & origen) Operador de asignación a partir de otra población.

Copia idénticamente las propiedades de otro objeto de la clase Poblacion.

Parámetros:

origen Objeto del que se hace copia

Devuelve:

Referencia a la poblacion

5.61.3.7. void ordenar (bool Maximizar = false) Ordena los miembros de la población segun su función de evaluación.

Parámetros:

Maximizar Si su valor es *true*, ordena a los individuos en forma descendente. De lo contrario los ordena de forma ascedente.

5.61.3.8. Individuo* remplazarIndividuo (Individuo * pNuevoInd, int pos)
[inline] Reemplaza el individuo ubicado en una posición determinada de la población por otro individuo.

Parámetros:

pNuevoInd Apuntador al idividuo que remplazará al individuo ubicado en la posición especificada.

pos Posición del individuo a remplazar.

Devuelve:

Apuntador al individuo reemplazado.

5.61.3.9. void SetAG ([AlgoritmoGenetico](#) * **Ag) [inline]** Cambia el algoritmo genetico al que pertenece la población.

Parámetros:

Ag Apuntador al objeto AlgoritmoGenético del que hará parte la poblacion.

5.61.3.10. int setTam (int *nuevoTam*, bool *reemplazar* = false, bool *crearAleatorios* = false) Cambia el tamaño de la población.

Si el nuevo tamaño es menor que el anterior, se eliminan los individuos restantes.
Si es mayor, los nuevos individuos son creados aleatoriamente dependiendo del valor de *crearAleatorios*

Parámetros:

nuevoTam Nuevo tamaño del arreglo.

crearAleatorios Si su valor es *true* se crean individuos aleatorios. Si es *false* se crean individuos con valores iniciales preestablecidos

reemplazar Si su valor es *true*, los individuos existentes en la población son reemplazados por nuevos individuos.

Devuelve:

Tamaño asignado o -1 en caso de error.

5.61.4. Documentación de los datos miembro

5.61.4.1. [AlgoritmoGenetico](#)* *m_pAG* [protected] Apuntador al objeto de la clase [AlgoritmoGenetico](#) al que pertenece la poblacion.

5.61.4.2. [Arreglo<Individuo>* m_pGeneracion](#) [protected] [Arreglo](#) de objetos de la clase [Individuo](#) que conforman la población.

La documentación para esta clase fué generada a partir de los siguientes archivos:

- [genetico.h](#)
- [genetico.cpp](#)

Capítulo 6

UNGenético Documentación de archivos

6.1. Referencia del Archivo arreglos.h

```
#include <string.h>
```

Clases

- class [Arreglo](#)

Clase genérica que almacena un arreglo de apuntadores a la clase T.

6.2. Referencia del Archivo genarreglo.h

```
#include "genbool.h"
#include "genentero.h"
#include "genreal.h"
```

Clases

- class [GenArreglo](#)

Clase derivada de la clase [Gen](#), especializada en un gen de tipo arreglo de tamaño variable.

- class [OperadorMutacionArreglo](#)

Clase derivada de la clase [OperadorMutacion](#) empleada en genes de tipo arreglo. G puede ser [GenBool](#), [GenEntero](#) o [GenReal](#). T puede ser bool, long o double.

- class [OperadorCruceArreglo](#)

Clase derivada de la clase [OperadorCruce](#) usada en genes de tipo arreglo. G puede ser [GenBool](#), [GenEntero](#) o [GenReal](#). T puede ser bool, long o double.

Definiciones

- #define [ADICIONAR_GENARREGLO_BOOL](#)(pIndividuo, pos, var, tamMin, tamMax)

Macro que adiciona un objeto [GenArregloBool](#) a un individuo.

- #define `ADICIONAR_GENARREGLO_ENTERO`(pIndividuo, pos, var, tamMin, tamMax, valMin, valMax, valInicial)

Macro que adiciona un objeto GenArregloEntero a un individuo.

- #define `ADICIONAR_GENARREGLO_REAL`(pIndividuo, pos, var, tamMin, tamMax, valMin, valMax, valInicial)

Macro que adiciona un objeto GenArregloReal a un individuo.

Tipos definidos

- typedef `Arreglo`< bool > `ArregloBool`

Define un arreglo de datos de tipo booleano (bool).

- typedef `Arreglo`< long > `ArregloEntero`

Define un arreglo de datos de tipo entero (long).

- typedef `Arreglo`< double > `ArregloReal`

Define un arreglo de datos de tipo real (double).

- typedef `GenArreglo`< `GenBool`, bool > `GenArregloBool`

Define un gen de tipo arreglo con datos de tipo booleano (bool).

- typedef `GenArreglo`< `GenEntero`, long > `GenArregloEntero`

Define un gen de tipo arreglo con datos de tipo entero (long).

- typedef `GenArreglo`< `GenReal`, double > `GenArregloReal`

Define un gen de tipo arreglo con datos de tipo real (double).

- typedef `OperadorMutacionArreglo< GenBool, bool > OperadorMutacionArregloBool`

Define un operador de mutación para genes de tipo arreglo con datos de tipo booleano (bool).

- typedef `OperadorMutacionArreglo< GenEntero, long > OperadorMutacionArregloEntero`

Define un operador de mutación para genes de tipo arreglo con datos de tipo entero (long).

- typedef `OperadorMutacionArreglo< GenReal, double > OperadorMutacionArregloReal`

Define un operador de mutación para genes de tipo arreglo con datos de tipo real (double).

- typedef `OperadorCruceArreglo< GenBool, bool > OperadorCruceArregloBool`

Define un operador de cruce para genes de tipo arreglo con datos de tipo booleano (bool).

- typedef `OperadorCruceArreglo< GenEntero, long > OperadorCruceArregloEntero`

Define un operador de cruce para genes de tipo arreglo con datos de tipo entero (long).

- typedef `OperadorCruceArreglo< GenReal, double > OperadorCruceArregloReal`

Define un operador de cruce para genes de tipo arreglo con datos de tipo real (double).

6.2.1. Documentación de las definiciones

6.2.1.1. #define ADICIONAR_GENARREGLO_BOOL(*pIndividuo*, *pos*, *var*, *tamMin*, *tamMax*) Macro que adiciona un objeto GenArregloBool a un individuo.

Se debe utilizar solamente en la función `codificacion()` de la clase derivada de [AlgoritmoGenetico](#). Su propósito particular es insertar un objeto GenArregloBool en una posición determinada del individuo apuntado por *pIndividuo*. Este objeto estará asociado con una de las variables del sistema a optimizar, perteneciente a la clase derivada de [AlgoritmoGenetico](#), para la cual se establecen el tamaño mínimo y tamaño máximo.

Parámetros:

pIndividuo Apuntador al individuo donde se adiciona el gen.

pos Posición en la que se adiciona el gen.

var Nombre de la variable del sistema a optimizar asociada con el gen adicionado. Debe ser de tipo ArregloBool,

tamMin Tamaño mínimo que puede tomar el gen.

tamMax Tamaño máximo que puede tomar el gen.

6.2.1.2. #define ADICIONAR_GENARREGLO_ENTERO(*pIndividuo*, *pos*, *var*, *tamMin*, *tamMax*, *valMin*, *valMax*, *valInicial*) Macro que adiciona un objeto GenArregloEntero a un individuo.

Se debe utilizar solamente en la función `codificacion()` de la clase derivada de [AlgoritmoGenetico](#). Su propósito particular es insertar un objeto GenArregloEntero en una posición determinada del individuo apuntado por *pIndividuo*. Este objeto estará asociado con una de las variables del sistema a optimizar, perteneciente a la clase derivada de [AlgoritmoGenetico](#), para la cual se establecen el tamaño mínimo, tamaño máximo y los valores mínimo, máximo e inicial.

Parámetros:

pIndividuo Apuntador al individuo donde se adiciona el gen.

pos Posición en la que se adiciona el gen.

var Nombre de la variable del sistema a optimizar asociada con el gen adicionado. Debe ser de tipo ArregloEntero,

tamMin Tamaño mínimo que puede tomar el gen.

tamMax Tamaño máximo que puede tomar el gen.

valMin Valor mínimo que puede tomar cada gen del arreglo.

valMax Valor máximo que puede tomar cada gen del arreglo.

valInicial Valor inicial que toma cada gen del arreglo.

6.2.1.3. #define ADICIONAR_GENARREGLO_REAL(pIndividuo, pos, var, tamMin, tamMax, valMin, valMax, valInicial) Macro que adiciona un objeto GenArregloReal a un individuo.

Se debe utilizar solamente en la función *codificacion()* de la clase derivada de [AlgoritmoGenetico](#). Su propósito particular es insertar un objeto GenArregloReal en una posición determinada del individuo apuntado por *pIndividuo*. Este objeto estará asociado con una de las variables del sistema a optimizar, perteneciente a la clase derivada de [AlgoritmoGenetico](#), para la cual se establecen el tamaño mínimo, tamaño máximo y los valores mínimo, máximo e inicial.

Parámetros:

pIndividuo Apuntador al individuo donde se adiciona el gen.

pos Posición en la que se adiciona el gen.

var Nombre de la variable del sistema a optimizar asociada con el gen adicionado. Debe ser de tipo ArregloReal,

tamMin Tamaño mínimo que puede tomar el gen.

tamMax Tamaño máximo que puede tomar el gen.

valMin Valor mínimo que puede tomar cada gen del arreglo.

valMax Valor máximo que puede tomar cada gen del arreglo.

valInicial Valor inicial que toma cada gen del arreglo.

6.2.2. Documentación de los tipos definidos

6.2.2.1. typedef *Arreglo*<bool> *ArregloBool* Define un arreglo de datos de tipo booleano (bool).

6.2.2.2. typedef *Arreglo*<long> *ArregloEntero* Define un arreglo de datos de tipo entero (long).

6.2.2.3. typedef *Arreglo*<double> *ArregloReal* Define un arreglo de datos de tipo real (double).

6.2.2.4. typedef *GenArreglo*<*GenBool*,bool> *GenArregloBool* Define un gen de tipo arreglo con datos de tipo booleano (bool).

6.2.2.5. typedef *GenArreglo*<*GenEntero*,long> *GenArregloEntero* Define un gen de tipo arreglo con datos de tipo entero (long).

6.2.2.6. typedef *GenArreglo*<*GenReal*,double> *GenArregloReal* Define un gen de tipo arreglo con datos de tipo real (double).

6.2.2.7. typedef *OperadorCruceArreglo*<*GenBool*,bool> *OperadorCruceArregloBool* Define un operador de cruce para genes de tipo arreglo con datos de tipo booleano (bool).

6.2.2.8. typedef `OperadorCruceArreglo<GenEntero,long>` `OperadorCruceArregloEntero` Define un operador de cruce para genes de tipo arreglo con datos de tipo entero (long).

6.2.2.9. typedef `OperadorCruceArreglo<GenReal,double>` `OperadorCruceArregloReal` Define un operador de cruce para genes de tipo arreglo con datos de tipo real (double).

6.2.2.10. typedef `OperadorMutacionArreglo<GenBool,bool>` `OperadorMutacionArregloBool` Define un operador de mutación para genes de tipo arreglo con datos de tipo booleano (bool).

6.2.2.11. typedef `OperadorMutacionArreglo<GenEntero,long>` `OperadorMutacionArregloEntero` Define un operador de mutación para genes de tipo arreglo con datos de tipo entero (long).

6.2.2.12. typedef `OperadorMutacionArreglo<GenReal,double>` `OperadorMutacionArregloReal` Define un operador de mutación para genes de tipo arreglo con datos de tipo real (double).

6.3. Referencia del Archivo genbool.h

```
#include "genetico.h"
```

Clases

- class [GenBool](#)

Clase derivada de la clase [Gen](#) especializada en un gen de tipo booleano.

- class [OperadorMutacionBoolUniforme](#)

Clase derivada de la clase [OperadorMutacion](#) que efectúa una mutación uniforme sobre un gen de tipo booleano.

- class [OperadorCruceBoolDiscreto](#)

Clase derivada de la clase [OperadorCruce](#) que define el cruce discreto entre dos genes de tipo booleano.

Definiciones

- #define [ADICIONAR_GENBOOL](#)(pIndividuo, pos, var, vallnicial)

Macro que adiciona un gen de tipo booleano al individuo.

6.3.1. Documentación de las definiciones

6.3.1.1. #define ADICIONAR_GENBOOL(pIndividuo, pos, var, vallnicial)

Macro que adiciona un gen de tipo booleano al individuo.

Se debe utilizar solamente en la función *codificacion()* de la clase derivada de [AlgoritmoGenetico](#). Tiene el propósito particular de insertar un objeto [GenBool](#) en una posición fija del individuo apuntado por *pIndividuo*. Este objeto estará asociado con una de las variables del sistema a optimizar perteneciente a la clase derivada de [AlgoritmoGenetico](#) de la cual se establece su valor inicial.

Parámetros:

pIndividuo Apuntador al individuo donde se adiciona el gen.

pos Posición en la que se adiciona el gen.

var Nombre de la variable del sistema a optimizar asociada con el gen adicionado.

valInicial Valor inicial que toma el gen.

6.4. Referencia del Archivo `genentero.h`

```
#include "genetico.h"
```

Clases

- class `GenEntero`

Clase derivada de la clase `Gen` especializada en un gen de tipo entero.

- class `OperadorMutacionEnteroUniforme`

Clase derivada de la clase `OperadorMutacion` que define una mutación uniforme sobre un gen de tipo entero.

- class `OperadorMutacionEnteroNoUniforme`

Clase derivada de la clase `OperadorMutacion` que define una mutación no uniforme sobre un gen de tipo entero.

- class `OperadorMutacionEnteroMuhlenbein`

Clase derivada de la clase `OperadorMutacion` que define una mutación Muhlenbein sobre un gen de de tipo entero.

- class `OperadorCruceEnteroPlano`

Clase derivada de la clase `OperadorCruce` que define un cruce plano entre dos genes de tipo entero.

- class `OperadorCruceEnteroAritmetico`

Clase derivada de la clase `OperadorCruce` que define un cruce aritmético entre dos genes de tipo entero.

- class [OperadorCruceEnteroBLX](#)

Clase derivada de la clase [OperadorCruce](#) que define un cruce BLX $-\alpha$ entre dos genes de tipo entero.

- class [OperadorCruceEnteroLineal](#)

Clase derivada de la clase [OperadorCruce](#) que define un cruce lineal entre dos genes de tipo entero.

- class [OperadorCruceEnteroDiscreto](#)

Clase derivada de la clase [OperadorCruce](#) que define un cruce discreto entre dos genes de tipo entero.

- class [OperadorCruceEnteroIntermedioExtendido](#)

Clase derivada de la clase [OperadorCruce](#) que define un cruce extendido intermedio entre dos genes de tipo entero.

- class [OperadorCruceEnteroHeuristico](#)

Clase derivada de la clase [OperadorCruce](#) que define un cruce heurístico entre dos genes de tipo entero.

- class [OperadorCruceEnteroLinealBGA](#)

Clase derivada de la clase [OperadorCruce](#) que define un cruce BGA lineal entre dos genes de tipo entero.

Definiciones

- #define [ADICIONAR_GENENTERO](#)(pIndividuo, pos, var, valMin, valMax, valInicial)

Macro que adiciona un gen de tipo entero a un individuo.

6.4.1. Documentación de las definiciones

6.4.1.1. #define ADICIONAR_GENENTERO(*pIndividuo*, *pos*, *var*, *valMin*, *valMax*, *valInicial*) Macro que adiciona un gen de tipo entero a un individuo.

Se debe utilizar solamente en la función *codificacion()* de la clase derivada de [AlgoritmoGenetico](#). Su propósito particular es insertar un objeto [GenEntero](#) en una posición determinada del individuo apuntado por *pIndividuo*. Este objeto estará asociado con una de las variables del sistema a optimizar, perteneciente a la clase derivada de [AlgoritmoGenetico](#). De esta variable se establecen los valores mínimo, máximo e inicial.

Parámetros:

pIndividuo Apuntador al individuo donde se adiciona el gen.

pos Posición en la que se adiciona el gen.

var Nombre de la variable del sistema a optimizar asociada con el gen adicionado.

valMin Valor mínimo que puede tomar el gen.

valMax Valor máximo que puede tomar el gen.

valInicial Valor inicial que toma el gen.

6.5. Referencia del Archivo genetico.h

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <math.h>
#include <float.h>
#include <time.h>
#include "arreglos.h"
```

Clases

- class [Gen](#)

Clase abstracta que sirve de base a clases que definen genes de distintos tipos.

- class [Individuo](#)

Clase que administra la información genética de un individuo.

- class [Poblacion](#)

Clase que administra la información de los individuos de un algoritmo genético.

- class [AlgoritmoGenetico](#)

Clase abstracta que administra el proceso de un algoritmo genético.

- class [OperadorAdaptacion](#)

Clase abstracta que define el proceso de adaptación del algoritmo genético.

- class [OperadorMutacion](#)

Clase abstracta que administra el proceso de mutación en el algoritmo genético.

- class [OperadorCruce](#)

Clase abstracta que administra el proceso de cruce entre los individuos del algoritmo genético.

- class [OperadorProbabilidad](#)

Clase abstracta que administra la asignación de la probabilidad de supervivencia a los individuos de la población.

- class [OperadorSeleccion](#)

Clase abstracta que administra el proceso de selección de individuos en la población.

- class [OperadorParejas](#)

Clase abstracta que administra la asignación de parejas para cada individuo de la población.

- class [OperadorReproduccion](#)

Clase abstracta que define la estrategia general de reproducción del algoritmo.

- class [OperadorFinalizacion](#)

Clase abstracta que define la estrategia de finalización del algoritmo genético.

- class [OperadorAdaptacionElitismo](#)

Clase derivada de la clase [OperadorAdaptacion](#) encargada de efectuar el proceso de elitismo para el algoritmo.

- class [OperadorAdaptacionProbMutacion](#)

Clase derivada de la clase [OperadorAdaptacion](#) que define la estrategia de adaptación para la probabilidad de mutación de los genes de cada individuo de la población.

- class [OperadorAdaptacionNumIndividuos](#)

Clase derivada de la clase [OperadorAdaptacion](#) que hace una variación del número de individuos del algoritmo genético.

- class [OperadorProbabilidadProporcional](#)

Clase derivada de la clase [OperadorProbabilidad](#) que define el proceso de asignación de probabilidad de supervivencia proporcional a cada individuo de la población.

- class [OperadorProbabilidadLineal](#)

Clase derivada de la clase [OperadorProbabilidad](#) que define el proceso de asignación de probabilidad de supervivencia lineal.

- class [OperadorProbabilidadHomogenea](#)

Clase derivada de la clase [OperadorProbabilidad](#) que define el proceso de asignación de probabilidad de supervivencia homogénea.

- class [OperadorSeleccionEstocasticaRemplazo](#)

Clase derivada de la clase [OperadorSeleccion](#) que define el proceso de selección estocástica con reemplazo de los individuos.

- class [OperadorParejasAleatorias](#)

Clase derivada de [OperadorParejas](#) que define la asignación de parejas aleatorias para los individuos de la población.

- class [OperadorParejasAdyacentes](#)

Clase derivada de la clase [OperadorParejas](#) que define la asignación de parejas adyacentes para los individuos de la población.

- class [OperadorParejasExtremos](#)

Clase derivada de la clase [OperadorParejas](#) que define la asignación de parejas extremas para los individuos de la población.

- class [OperadorReproduccionCruceSimple](#)

Clase derivada de la clase [OperadorReproduccion](#) que define el cruce simple entre dos individuos de la población.

- class [OperadorReproduccionDosPadresDosHijos](#)

Clase derivada de la clase [OperadorReproduccion](#) que define la estrategia de reproducción dos padres dos hijos.

- class [OperadorReproduccionMejorPadreMejorHijo](#)

Clase derivada de la clase [OperadorReproduccion](#) que define la estrategia de reproducción mejor padre mejor hijo.

- class [OperadorReproduccionMejoresEntrePadresEHijos](#)

Clase derivada de la clase [OperadorReproduccion](#) que define la estrategia de reproducción mejores entre padres e hijos.

- class [OperadorFinalizacionOnline](#)

Clase derivada de la clase [OperadorFinalizacion](#) que define la finalización del algoritmo basándose en su medida online.

- class [OperadorFinalizacionOffline](#)

Clase derivada de la clase [OperadorFinalizacion](#) que define la finalización del algoritmo basándose en su medida online.

Definiciones

- #define [DECLARAR_ALGORITMO](#)(nombreAlgoritmo)
Macro que crea una nueva clase derivada de [AlgoritmoGenetico](#).
- #define [FIN_DECLARAR_ALGORITMO](#)
Macro que termina la creación de una clase derivada de [AlgoritmoGenetico](#).
- #define [DEFINIR_OPERADOR_PROBABILIDAD](#)(tipoOperador)
Macro que define el operador de probabilidad que se empleará en el algoritmo genético.
- #define [DEFINIR_OPERADOR_SELECCION](#)(tipoOperador)
Macro que define el operador de selección que se empleará en el algoritmo genético.
- #define [DEFINIR_OPERADOR_PAREJAS](#)(tipoOperador)
Macro que define el operador de parejas que se empleará en el algoritmo genético.
- #define [DEFINIR_OPERADOR_REPRODUCCION](#)(tipoOperador)
Macro que define el operador de reproducción que se empleará en el algoritmo genético.
- #define [ADICIONAR_OPERADOR_ADAPTACION](#)(tipoOperador)

Macro que define los operadores de adaptación que se emplearán en el algoritmo genético.

- #define `ADICIONAR_OPERADOR_FINALIZACION`(tipoOperador)

Macro que define los operadores de finalización que se emplearán en el algoritmo genético.

- #define `ADICIONAR_OPERADOR_MUTACION`(tipoOperador)

Macro que define el operador de mutación que usará el algoritmo genético para un gen en particular.

- #define `ADICIONAR_OPERADOR_CRUCE`(tipoOperador)

Macro que define el operador de cruce que usará el algoritmo genético para un gen en particular.

Enumeraciones

- enum `CriteriosDeAdaptacion` { `ADAPTACION_PROB_MUTACION_OFFLINE` = 1, `ADAPTACION_PROB_MUTACION_EXPONENCIAL` }

Constantes utilizadas por los operadores de adaptación.

- enum `EstadosIndividuo` {
`ESTADO_CODIFICAR` = 1, `ESTADO_DECODIFICAR`,
`ESTADO_CREAR` }

Constantes utilizadas en la definición de la codificación del individuo.

Funciones

- long [redondear](#) (double)
- template<class T> T [restringir](#) (T valor, T min, T max)

6.5.1. Documentación de las definiciones

6.5.1.1. **#define ADICIONAR_OPERADOR_ADAPTACION(tipoOperador)**

Macro que define los operadores de adaptación que se emplearán en el algoritmo genético.

Parámetros:

tipoOperador Especifica el tipo de operador a utilizar junto con sus parámetros iniciales.

6.5.1.2. **#define ADICIONAR_OPERADOR_CRUCE(tipoOperador)** Macro

que define el operador de cruce que usará el algoritmo genético para un gen en particular.

Parámetros:

tipoOperador Especifica el tipo de operador a utilizar junto con sus parámetros iniciales.

6.5.1.3. **#define ADICIONAR_OPERADOR_FINALIZACION(tipoOperador)**

Macro que define los operadores de finalización que se emplearán en el algoritmo genético.

Parámetros:

tipoOperador Especifica el tipo de operador a utilizar junto con sus parámetros iniciales.

6.5.1.4. #define ADICIONAR_OPERADOR_MUTACION(tipoOperador) Macro que define el operador de mutación que usará el algoritmo genético para un gen en particular.

Parámetros:

tipoOperador Especifica el tipo de operador a utilizar junto con sus parámetros iniciales.

6.5.1.5. #define DECLARAR_ALGORITMO(nombreAlgoritmo) Macro que crea una nueva clase derivada de [AlgoritmoGenetico](#).

Declara e implementa el constructor por defecto de esta clase, también declara implícitamente las funciones de carácter obligatorio, las cuales deben ser implementadas en el proyecto de optimización, usando la forma:

```
tipoRetornado nombreAlgoritmo::nombreFuncion(parametros)
{
    //cuerpo de la función
}
```

Por ejemplo:

```
double nombreAlgoritmo::objetivo()
{
    double FO;
    //instrucciones correspondientes a la funcion objetivo
    return FO;
}
```

Pueden declararse e implementarse adicionalmente otros constructores, métodos y variables. También puede adicionarse un destructor, en caso de necesitar destruir apuntadores a objetos creados dinámicamente.

6.5.1.6. #define DEFINIR_OPERADOR_PAREJAS(tipoOperador) Macro que define el operador de parejas que se empleará en el algoritmo genético.

Parámetros:

tipoOperador Especifica el tipo de operador a utilizar junto con sus parámetros iniciales.

6.5.1.7. #define DEFINIR_OPERADOR_PROBABILIDAD(tipoOperador) Macro que define el operador de probabilidad que se empleará en el algoritmo genético.

Parámetros:

tipoOperador Especifica el tipo de operador a utilizar junto con sus parámetros iniciales.

6.5.1.8. #define DEFINIR_OPERADOR_REPRODUCCION(tipoOperador) Macro que define el operador de reproducción que se empleará en el algoritmo genético.

Parámetros:

tipoOperador Especifica el tipo de operador a utilizar junto con sus parámetros iniciales.

6.5.1.9. #define DEFINIR_OPERADOR_SELECCION(tipoOperador) Macro que define el operador de selección que se empleará en el algoritmo genético.

Parámetros:

tipoOperador Especifica el tipo de operador a utilizar junto con sus parámetros iniciales.

6.5.1.10. #define FIN_DECLARAR_ALGORITMO Macro que termina la creación de una clase derivada de [AlgoritmoGenetico](#).

6.5.2. Documentación de las enumeraciones

6.5.2.1. enum CriteriosDeAdaptacion Constantes utilizadas por los operadores de adaptación.

Valores de la enumeración:

ADAPTACION_PROBMUTACION_OFFLINE

ADAPTACION_PROBMUTACION_EXPONENCIAL

6.5.2.2. enum EstadosIndividuo Constantes utilizadas en la definicion de la codificación del individuo.

Se utilizan en la función *codificacion()* de la clase [AlgoritmoGenetico](#)

Valores de la enumeración:

ESTADO_CODIFICAR

ESTADO_DECODIFICAR

ESTADO_CREAR

6.5.3. Documentación de las funciones

6.5.3.1. long redondear (double x) Retorna el entero mas cercano a x

Parámetros:

x Número del que se desea encontrar su valor entero más cercano

Devuelve:

Número entero más cercano a x

6.5.3.2. T restringir (T *valor*, T *min*, T *max*) [*inline*] Comprueba si *valor* se encuentra dentro de un rango determinado y lo restringe a los límites.

Parámetros:

valor Valor a restringir

min Límite inferior del rango

max Límite superior del rango

Devuelve:

Valor ubicado dentro de los límites. Si $\text{valor} < \text{min}$ retorna min . Si $\text{valor} > \text{max}$ retorna max

6.6. Referencia del Archivo genreal.h

```
#include "genetico.h"
```

Clases

- class [GenReal](#)

Clase derivada de la clase [Gen](#) especializada en un gen de tipo real.

- class [OperadorMutacionRealUniforme](#)

Clase derivada de la clase [OperadorMutacion](#) que define una mutación uniforme sobre un gen de tipo real.

- class [OperadorMutacionRealNoUniforme](#)

Clase derivada de la clase [OperadorMutacion](#) que define una mutación no uniforme sobre un gen de tipo real.

- class [OperadorMutacionRealMuhlenbein](#)

Clase derivada de la clase [OperadorMutacion](#) que define una mutación Muhlenbein sobre un gen de de tipo real.

- class [OperadorCruceRealPlano](#)

Clase derivada de la clase [OperadorCruce](#) que define un cruce plano entre dos genes de tipo real.

- class [OperadorCruceRealAritmetico](#)

Clase derivada de la clase [OperadorCruce](#) que define un cruce aritmético entre dos genes de tipo real.

- class [OperadorCruceRealBLX](#)

Clase derivada de la clase [OperadorCruce](#) que define un cruce BLX - α entre dos genes de tipo real.

- class [OperadorCruceRealLineal](#)

Clase derivada de la clase [OperadorCruce](#) que define un cruce lineal entre dos genes de tipo real.

- class [OperadorCruceRealDiscreto](#)

Clase derivada de la clase [OperadorCruce](#) que define un cruce discreto entre dos genes de tipo real.

- class [OperadorCruceRealIntermedioExtendido](#)

Clase derivada de la clase [OperadorCruce](#) que define el cruce extendido intermedio entre dos genes de tipo real.

- class [OperadorCruceRealHeuristico](#)

Clase derivada de la clase [OperadorCruce](#) que define un cruce heurístico entre dos genes de tipo real.

- class [OperadorCruceRealLinealBGA](#)

Clase derivada de la clase [OperadorCruce](#) que define un cruce BGA lineal entre genes de tipo real.

Definiciones

- #define [ADICIONAR_GENREAL](#)(plIndividuo, pos, var, valMin, valMax, valInicial)

Macro que adiciona un gen de tipo real al individuo.

6.6.1. Documentación de las definiciones

6.6.1.1. #define ADICIONAR_GENREAL(*pIndividuo*, *pos*, *var*, *valMin*, *valMax*, *valInicial*) Macro que adiciona un gen de tipo real al individuo.

Se debe utilizar solamente en la función *codificacion()* de la clase derivada de [AlgoritmoGenetico](#). Su propósito particular es insertar un objeto [GenReal](#) en una posición determinada del individuo apuntado por *pIndividuo*. Este objeto estará asociado con una de las variables del sistema a optimizar, perteneciente a la clase derivada de [AlgoritmoGenetico](#), para la cual se establecen los valores mínimo, máximo e inicial.

Parámetros:

pIndividuo Apuntador al individuo donde se adiciona el gen.

pos Posición en la que se adiciona el gen.

var Nombre de la variable del sistema a optimizar asociada con el gen adicionado.

valMin Valor mínimo que puede tomar el gen.

valMax Valor máximo que puede tomar el gen.

valInicial Valor inicial que toma el gen.

6.7. Referencia del Archivo UNGenetico.h

```
#include "genetico.cpp"  
#include "genarreglo.h"  
#include "genbool.cpp"  
#include "genentero.cpp"  
#include "genreal.cpp"
```

6.8. Referencia del Archivo ventana.h

```
#include "wx/wxprec.h"
#include "wx/wx.h"
#include "wx/colordlg.h"
#include <wx/notebook.h>
#include <wx/panel.h>
#include <wx/spinctrl.h>
#include <wx/textfile.h>
#include <wx/image.h>
#include <time.h>
#include <stdlib.h>
```

Clases

- class [AGVentana](#)

Clase derivada de wxScrolledWindow de la librería wxWindows que contiene objetos visibles por el usuario.

- class [AGFrame](#)

Clase derivada de wxFrame de la librería wxWindows que contiene las ventanas de la clase [AGVentana](#).

Definiciones

- #define [DECLARAR_APLICACION](#)(nombreApp)

Macro que crea una aplicación gráfica para UNGenético.

- `#define EJECUTAR_EVENTOS { while(wxGetApp().Pending()) wxGetApp().Dispatch(); }`

Ejecuta los eventos pendientes de la aplicación.

Enumeraciones

- `enum Paginas {
 ID_CONSOLA = 2000, ID_GRAFICA,
 ID_OTRA }`

6.8.1. Documentación de las definiciones

6.8.1.1. `#define DECLARAR_APLICACION(nombreApp)` Valor:

```
class nombreApp : public wxApp           \
{                                         \
public:                                   \
    /*Puntero al frame*/                 \
    AGFrame *m_pFrame;                   \
    /*Programa principal*/               \
    virtual bool OnInit();               \
};                                         \
IMPLEMENT_APP(nombreApp)
```

Macro que crea una aplicación gráfica para UNGenético.

Declara una clase derivada de *wxApp* que representa a la aplicación por si misma junto con un apuntador al objeto que contiene a las otras ventanas y sus controles; también declara el método *OnInit()* que corresponde a la función principal donde se invocan las instrucciones que desarrolla la aplicación. Por último usa la macro `IMPLEMENT_APP(nombreApp)` para crear una instancia de la clase derivada de *wxApp*.

Parámetros:

nombreApp nombre a dar a la clase para la aplicación, derivada de *wxApp*.

6.8.1.2. `#define EJECUTAR_EVENTOS { while(wxGetApp().Pending()) wxGetApp().Dispatch(); }` Ejecuta los eventos pendientes de la aplicación.

Debe usarse en medio de largos procesos para ejecutar eventos como la actualización de la visualización de la ventana

6.8.2. Documentación de las enumeraciones

6.8.2.1. enum [Paginas](#)

Valores de la enumeración:

ID_CONSOLA

ID_GRAFICA

ID_OTRA

Apéndice A

Valores por defecto de UNGenético

A.1. Valores por defecto de la clase Algoritmo-Genetico

Miembro	Valor
m_GeneracionMaxima	100
m_TamanoPoblacion	10
m_IndicadorUsarAdaptacion	true
m_IndicadorInicializarPoblacionAleatoria	true
m_IndicadorMaximizar	false
m_IndicadorArchivo	true
m_IntervaloSalvar	1
m_NombreArchivo	"salidas.txt"
m_IndicadorMostrar	false
m_IndicadorMostrarMejorEnHistoria	true
m_IndicadorMostrarGeneracionMejorHistorico	true
m_IndicadorMostrarMejorEnGeneracion	true
m_IndicadorMostrarPeorEnGeneracion	true

m_IndicadorMostrarMedia	true
m_IndicadorMostrarDesviacion	true
m_IndicadorMostrarOnLine	true
m_IndicadorMostrarOffLine	true

A.2. Operadores por defecto de UNGenético

Operador	Operador por defecto
OperadorProbabilidad	OperadorProbabilidadLineal
OperadorSeleccion	OperadorSeleccionEstocasticaRemplazo
OperadorParejas	OperadorParejasAdyacentes
OperadorReproduccion	OperadorReproduccionMejorPadreMejorHijo
OperadorAdaptacion	OperadorAdaptacionElitismo
OperadorFinalizacion	

Tipo de Gen	Operador	Operador por defecto
GenBool	OperadorMutacion	OperadorMutacionBoolUniforme
	OperadorCruce	OperadorCruceBoolDiscreto
GenEntero	OperadorMutacion	OperadorMutacionEnteroUniforme
	OperadorCruce	OperadorCruceEnteroPlano
GenReal	OperadorMutacion	OperadorMutacionRealUniforme
	OperadorCruce	OperadorCruceRealBLX
GenArreglo	OperadorMutacion	OperadorMutacionArreglo
	OperadorCruce	OperadorCruceArreglo

NOTA: NO hay un operador de finalización establecido por defecto, el algoritmo genético termina su ejecución al completar el número máximo de generaciones establecido. Cuando se definen otros operadores de finalización como [Operador-FinalizacionOffline](#) y [OperadorFinalizacionOnline](#) el algoritmo termina su ejecución al recibir la primera orden de parada de cualquiera de estos operadores o al completar el número máximo de iteraciones.

A.3. Operadores de UNGenético

	Operador	Parámetros por defecto
OperadorProbabilidad	OperadorProbabilidadProporcional	
	OperadorProbabilidadLineal	nmin=0.5
	OperadorProbabilidadHomogenea	
OperadorSeleccion	OperadorSeleccionEstocasticaRemplazo	
OperadorParejas	OperadorParejasAleatoria	
	OperadorParejasSiguierte	
	OperadorParejasExtremos	
OperadorReproduccion	OperadorReproduccionCruceSimple	
	OperadorReproduccionDosPadresDosHijos	
	OperadorReproduccionMejorPadreMejorHijo	
	OperadorReproduccionMejoresEntrePadresEHijos	
OperadorAdaptacion	OperadorAdaptacionElitismo	
	OperadorAdaptacionNumIndividuos	NumIndivInicio = TamañoPoblacion
		NumIndivFin = TamañoPoblacion/4
	OperadorAdaptacionProbMutacion ADAPTACION_PROBMUTACION_OFFLINE	MaxProb=0.5
		FactorVariacion=0.001
		Escalon = 0.1
	OperadorAdaptacionProbMutacion ADAPTACION_PROBMUTACION_EXPONENCIAL	MaxCont = 10
		MaxProb=0.5
OperadorFinalizacion	OperadorFinalizacionOnline	T = GeneraciónMaxima/2
	OperadorFinalizacionOffline	FactorVariacion=0.001
		MaxCont=30
GenBool	OperadorFinalizacionOnline	FactorVariacion=0.0005
	OperadorFinalizacionOffline	MaxCont=30
GenBool	OperadorMutacionBoolUniforme	
	OperadorCruceBoolPlano	
GenEntero	OperadorMutacionEnteroUniforme	

	OperadorMutacionEnteroNoUniforme	b=0.5
	OperadorMutacionEnteroMuhlenbein	Factor=0.1
	OperadorCruceEnteroPlano	
	OperadorCruceEnteroAritmetico	Lambda=0.7
	OperadorCruceEnteroBLX	Alfa=0.3
	OperadorCruceEnteroLineal	
	OperadorCruceEnteroDiscreto	
	OperadorCruceEnteroIntermedioExtendido	
	OperadorCruceEnteroHeuristico	
	OperadorCruceEnteroLinealBGA	
GenReal	OperadorMutacionRealUniforme	
	OperadorMutacionRealNoUniforme	b=0.5
	OperadorMutacionRealMuhlenbein	Factor=0.1
	OperadorCruceRealPlano	
	OperadorCruceRealAritmetico	Lambda=0.7
	OperadorCruceRealBLX	Alfa=0.3
	OperadorCruceRealLineal	
	OperadorCruceRealDiscreto	
	OperadorCruceRealIntermedioExtendido	
	OperadorCruceRealHeuristico	
	OperadorCruceRealLinealBGA	
GenArreglo	OperadorMutacionArreglo	
	OperadorCruceArreglo	

Apéndice B

GNU Library General Public License, Version 2

Copyright (C) 1991 Free Software Foundation, Inc. 675 Mass Ave, Cambridge, MA 02139, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the library GPL. It is numbered 2 because it goes with version 2 of the ordinary GPL.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software – to make sure the software is free for all its users.

This license, the Library General Public License, applies to some specially designated Free Software Foundation software, and to any other libraries whose authors decide to use it. You can use it for your libraries, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to

distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library, or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link a program with the library, you must provide complete object files to the recipients so that they can relink them with the library, after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the library.

Also, for each distributor's protection, we want to make certain that everyone understands that there is no warranty for this free library. If the library is modified by someone else and passed on, we want its recipients to know that what they have is not the original version, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that companies distributing free software will individually obtain patent licenses, thus in effect transforming the program into proprietary software. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License, which was designed for utility programs. This license, the

GNU Library General Public License, applies to certain designated libraries. This license is quite different from the ordinary one; be sure to read it in full, and don't assume that anything in it is the same as in the ordinary license.

The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is analogous to running a utility program or application program. However, in a textual and legal sense, the linked executable is a combined work, a derivative of the original library, and the ordinary General Public License treats it as such.

Because of this blurred distinction, using the ordinary General Public License for libraries did not effectively promote software sharing, because most developers did not use the libraries. We concluded that weaker conditions might promote sharing better.

However, unrestricted linking of non-free programs would deprive the users of those programs of all benefit from the free status of the libraries themselves. This Library General Public License is intended to permit developers of non-free programs to use free libraries, while preserving your freedom as a user of such programs to change the free libraries that are incorporated in them. (We have not seen how to achieve this as regards changes in header files, but we have achieved it as regards changes in the actual functions of the Library.) The hope is that this will lead to faster development of free libraries.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, while the latter only works together with the library.

Note that it is possible for a library to be covered by the ordinary General Public License rather than by this special one.

B.0.1. GNU LIBRARY GENERAL PUBLIC LICENSE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION 0. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of

warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library. b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But

when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated

place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also compile or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse

engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

- a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)
- b) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.
- c) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.
- d) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above. b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy,

distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who

places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Library General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY 15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LI-

BRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS Appendix: How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the library's name and a brief idea of what it does.> Copyright (C) <year> <name of author>

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

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public License for more details.

You should have received a copy of the GNU Library General Public License along with this library; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the library 'Frob' (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990 Ty Coon, President of Vice

That's all there is to it!

Índice alfabético

- ~AGFrame
 - AGFrame, [36](#)
- ~AGVentana
 - AGVentana, [46](#)
- ~AlgoritmoGenetico
 - AlgoritmoGenetico, [57](#)
- ~Arreglo
 - Arreglo, [73](#)
- ~Gen
 - Gen, [81](#)
- ~GenArreglo
 - GenArreglo, [89](#)
- ~GenBool
 - GenBool, [99](#)
- ~GenEntero
 - GenEntero, [105](#)
- ~GenReal
 - GenReal, [113](#)
- ~Individuo
 - Individuo, [121](#)
- ~OperadorAdaptacion
 - OperadorAdaptacion, [127](#)
- ~OperadorAdaptacionElitismo
 - OperadorAdaptacionElitismo, [129](#)
- ~OperadorAdaptacionNumIndividuos
 - OperadorAdaptacionNumIndividuos, [131](#)
- ~OperadorAdaptacionProbMutacion
 - OperadorAdaptacionProbMutacion, [136](#)
- ~OperadorCruce
 - OperadorCruce, [140](#)
- ~OperadorCruceArreglo
 - OperadorCruceArreglo, [143](#)
- ~OperadorCruceBoolDiscreto
 - OperadorCruceBoolDiscreto, [146](#)
- ~OperadorCruceEnteroAritmetico
 - OperadorCruceEnteroAritmetico, [148](#)
- ~OperadorCruceEnteroBLX
 - OperadorCruceEnteroBLX, [152](#)
- ~OperadorCruceEnteroDiscreto
 - OperadorCruceEnteroDiscreto, [156](#)

~OperadorCruceEnteroHeuristico OperadorCruceEnteroHeuristico, 158	~OperadorCruceRealLinealBGA OperadorCruceRealLinealBGA, 191
~OperadorCruceEnteroIntermedioExtendido OperadorCruceEntero- IntermedioExtendido, 161	~OperadorCruceRealPlano OperadorCruceRealPlano, 194
~OperadorCruceEnteroLineal OperadorCruceEnteroLineal, 164	~OperadorFinalizacion OperadorFinalizacion, 196
~OperadorCruceEnteroLinealBGA OperadorCruceEnteroLineal- BGA, 167	~OperadorFinalizacionOffline OperadorFinalizacionOffline, 199
~OperadorCruceEnteroPlano OperadorCruceEnteroPlano, 170	~OperadorFinalizacionOnline OperadorFinalizacionOnline, 203
~OperadorCruceRealAritmetico OperadorCruceRealAritmetico, 172	~OperadorMutacion OperadorMutacion, 207
~OperadorCruceRealBLX OperadorCruceRealBLX, 176	~OperadorMutacionArreglo OperadorMutacionArreglo, 211
~OperadorCruceRealDiscreto OperadorCruceRealDiscreto, 180	~OperadorMutacionBoolUniforme OperadorMutacionBoolUniforme, 214
~OperadorCruceRealHeuristico OperadorCruceRealHeuristico, 182	~OperadorMutacionEnteroMuhlenbein OperadorMutacionEntero- Muhlenbein, 219
~OperadorCruceRealIntermedioExtendido OperadorCruceRealIntermedio- Extendido, 185	~OperadorMutacionEnteroNoUniforme OperadorMutacionEnteroNo- Uniforme, 224
~OperadorCruceRealLineal OperadorCruceRealLineal, 188	~OperadorMutacionEnteroUniforme OperadorMutacionEntero- Uniforme, 229
	~OperadorMutacionRealMuhlenbein OperadorMutacionReal- Muhlenbein, 234

~OperadorMutacionRealNoUniforme
 OperadorMutacionRealNo-
 Uniforme, [239](#)

~OperadorMutacionRealUniforme
 OperadorMutacionRealUniforme,
 [244](#)

~OperadorParejas
 OperadorParejas, [248](#)

~OperadorParejasAdyacentes
 OperadorParejasAdyacentes,
 [250](#)

~OperadorParejasAleatorias
 OperadorParejasAleatorias, [252](#)

~OperadorParejasExtremos
 OperadorParejasExtremos, [254](#)

~OperadorProbabilidad
 OperadorProbabilidad, [256](#)

~OperadorProbabilidadHomogenea
 OperadorProbabilidad-
 Homogenea, [258](#)

~OperadorProbabilidadLineal
 OperadorProbabilidadLineal, [260](#)

~OperadorProbabilidadProporcional
 OperadorProbabilidad-
 Proporcional, [264](#)

~OperadorReproduccion
 OperadorReproduccion, [267](#)

~OperadorReproduccionCruceSimple
 OperadorReproduccionCruce-
 Simple, [269](#)

~OperadorReproduccionDosPadresDosHijos
 OperadorReproduccionDos-
 PadresDosHijos, [271](#)

~OperadorReproduccionMejorPadreMejorHijo
 OperadorReproduccionMejor-
 PadreMejorHijo, [275](#)

~OperadorReproduccionMejoresEntrePadresEHijos
 OperadorReproduccionMejores-
 EntrePadresEHijos, [273](#)

~OperadorSeleccion
 OperadorSeleccion, [277](#)

~OperadorSeleccionEstocasticaRemplazo
 OperadorSeleccionEstocastica-
 Remplazo, [279](#)

~Poblacion
 Poblacion, [282](#)

actualizarMedidas
 AlgoritmoGenetico, [57](#)

adaptacion
 AlgoritmoGenetico, [58](#)
 OperadorAdaptacion, [127](#)
 OperadorAdaptacionElitismo,
 [129](#)
 OperadorAdaptacionNum-
 Individuos, [132](#)
 OperadorAdaptacionProb-
 Mutacion, [136](#)

ADAPTACION_PROBMUTACION_-
 EXPONENCIAL
 genetico.h, [309](#)

ADAPTACION_PROBMUTACION_-
OFFLINE
genetico.h, [309](#)

Adicionar
Arreglo, [73](#)

ADICIONAR_GENARREGLO_-
BOOL
genarreglo.h, [291](#)

ADICIONAR_GENARREGLO_-
ENTERO
genarreglo.h, [291](#)

ADICIONAR_GENARREGLO_-
REAL
genarreglo.h, [292](#)

ADICIONAR_GENBOOL
genbool.h, [295](#)

ADICIONAR_GENENTERO
genentero.h, [299](#)

ADICIONAR_GENREAL
genreal.h, [313](#)

ADICIONAR_OPERADOR_-
ADAPTACION
genetico.h, [306](#)

ADICIONAR_OPERADOR_CRUCE
genetico.h, [306](#)

ADICIONAR_OPERADOR_-
FINALIZACION
genetico.h, [306](#)

ADICIONAR_OPERADOR_-
MUTACION
genetico.h, [306](#)

adicionarGen
Individuo, [121](#)

AGFrame, [31](#)
~AGFrame, [36](#)
AGFrame, [36](#)
AgregarPagina, [36](#)
m_congelar, [38](#)
m_grafica, [38](#)
m_NomArch, [39](#)
m_pCongelarMenu, [39](#)
m_pFileMenu, [39](#)
m_pInfoMenu, [39](#)
m_pMenuBar, [39](#)
m_pMenuGrafica, [39](#)
m_pNotebook, [39](#)
m_pPagina, [39](#)
m_pPanel, [39](#)
m_pSizerFrame, [39](#)
m_pSizerNB, [39](#)
m_pTextCtrl, [40](#)
MENU_CONGELAR, [36](#)
MENU_FILE_QUIT, [35](#)
MENU_FILE_SAVE, [35](#)
MENU_GRAFICA_FONDO, [35](#)
MENU_GRAFICA_GRAFCAR,
[36](#)
MENU_GRAFICA_GRILLA, [35](#)
MENU_GRAFICA_LINEA, [35](#)
MENU_INFO_ABOUT, [35](#)

- menus, [35](#)
- NOTEBOOK, [36](#)
- ObtenerPagina, [36](#)
- OnClose, [37](#)
- OnCongelar, [37](#)
- OnMenuFileQuit, [37](#)
- OnMenuFileSave, [37](#)
- OnMenuFondo, [37](#)
- OnMenuGrafica, [37](#)
- OnMenuGrilla, [37](#)
- OnMenuInfoAbout, [38](#)
- OnMenuLinea, [38](#)
- SelColor, [38](#)
- SetPagina, [38](#)
- AgregarPagina
 - AGFrame, [36](#)
- AGVentana, [41](#)
 - ~AGVentana, [46](#)
 - AGVentana, [46](#)
 - Controles, [45](#)
 - Extremo, [46](#)
 - GrafGrilla, [46](#)
 - GrafTexto, [46](#)
 - m_alto, [48](#)
 - m_ancho, [48](#)
 - m_ColBrush, [48](#)
 - m_ColGraf, [48](#)
 - m_ColGrilla, [48](#)
 - m_Columna, [48](#)
 - m_dib, [48](#)
 - m_dibCopia, [48](#)
 - m_Ini, [48](#)
 - m_max, [48](#)
 - m_min, [48](#)
 - m_NameArch, [48](#)
 - m_NumGen, [49](#)
 - m_pBitmap, [49](#)
 - m_pSizer, [49](#)
 - m_pTextarch, [49](#)
 - m_pTxtConsola, [49](#)
 - m_pValores, [49](#)
 - m_tam, [49](#)
 - m_Titulo, [49](#)
 - ObtenerExtremos, [47](#)
 - ObtenerFrame, [47](#)
 - ObtenerValoresGrafica, [47](#)
 - ObtenerY, [47](#)
 - OnMaxLen, [47](#)
 - OnPaint, [47](#)
 - OTRO, [45](#)
 - TEXT_CONSOLA, [45](#)
- AlgoritmoGenetico, [50](#)
 - AlgoritmoGenetico, [57](#)
- AlgoritmoGenetico
 - ~AlgoritmoGenetico, [57](#)
 - actualizarMedidas, [57](#)
 - adaptacion, [58](#)
 - AlgoritmoGenetico, [57](#)
 - asignarParejas, [58](#)
 - asignarProbabilidad, [58](#)

codificacion, [58](#)
 crearOperadores, [59](#)
 definirOperadores, [60](#)
 finalizar, [60](#)
 finalizarOptimizacion, [60](#)
 inicializarApuntadores, [60](#)
 inicializarParametros, [60](#)
 inicializarVariables, [60](#)
 iniciarOptimizacion, [61](#)
 iterarOptimizacion, [61](#)
 m_Desviacion, [64](#)
 m_Generacion, [64](#)
 m_GeneracionDelMejorEnLaHistoria, [64](#)
 m_GeneracionMaxima, [65](#)
 m_IndicadorArchivo, [65](#)
 m_IndicadorInicializarPoblacionAleatoria, [65](#)
 m_IndicadorMaximizar, [65](#)
 m_IndicadorMostrar, [65](#)
 m_IndicadorMostrarDesviacion, [65](#)
 m_IndicadorMostrarGeneracionMejorHistoria, [65](#)
 m_IndicadorMostrarMedia, [65](#)
 m_IndicadorMostrarMejorEnGeneracion, [65](#)
 m_IndicadorMostrarMejorEnHistoria, [66](#)
 m_IndicadorMostrarOffLine, [66](#)
 m_IndicadorMostrarOnLine, [66](#)
 m_IndicadorMostrarPeorEnGeneracion, [66](#)
 m_IndicadorUsarAdaptacion, [66](#)
 m_IntervaloSalvar, [66](#)
 m_Media, [66](#)
 m_MedidaOffLine, [66](#)
 m_MedidaOffLineAnterior, [66](#)
 m_MedidaOnLine, [67](#)
 m_MedidaOnLineAnterior, [67](#)
 m_NombreArchivo, [67](#)
 m_pListaOperadorAdaptacion, [67](#)
 m_pListaOperadorCruce, [67](#)
 m_pListaOperadorFinalizacion, [67](#)
 m_pListaOperadorMutacion, [67](#)
 m_pMejorEnEstaGeneracion, [67](#)
 m_pMejorEnLaHistoria, [68](#)
 m_pModelo, [68](#)
 m_pOpParejas, [68](#)
 m_pOpProbabilidad, [68](#)
 m_pOpReproduccion, [68](#)
 m_pOpSeleccion, [68](#)
 m_pPeorEnEstaGeneracion, [68](#)
 m_pPoblacionActual, [68](#)
 m_TamanoPoblacion, [68](#)
 mostrarMedidas, [61](#)
 mutar, [62](#)
 objetivo, [62](#)

- optimizar, [63](#)
- reproducir, [63](#)
- salvar, [63](#)
- seleccionar, [64](#)
- Arreglo, [70](#)
 - ~Arreglo, [73](#)
 - Adicionar, [73](#)
 - Arreglo, [73](#)
 - asignarMemoria, [74](#)
 - Destroy, [74](#)
 - Detach, [74](#)
 - FlushDestroy, [75](#)
 - FlushDetach, [75](#)
 - getObj, [75](#)
 - getPtr, [75](#)
 - getSize, [76](#)
 - Insertar, [76](#)
 - IntercambiarPos, [76](#)
 - liberarMemoria, [77](#)
 - m_capacidad, [79](#)
 - m_items, [79](#)
 - m_pData, [79](#)
 - operator=, [77](#)
 - operator[], [77](#)
 - reemplazar, [78](#)
 - Truncar, [78](#)
- ArregloBool
 - genarreglo.h, [293](#)
- ArregloEntero
 - genarreglo.h, [293](#)
- ArregloReal
 - genarreglo.h, [293](#)
- arreglos.h, [287](#)
- asignarMemoria
 - Arreglo, [74](#)
- asignarPareja
 - Individuo, [121](#)
- asignarParejas
 - AlgoritmoGenetico, [58](#)
 - OperadorParejas, [248](#)
 - OperadorParejasAdyacentes, [250](#)
 - OperadorParejasAleatorias, [252](#)
 - OperadorParejasExtremos, [254](#)
- asignarProbabilidad
 - AlgoritmoGenetico, [58](#)
 - Individuo, [122](#)
 - OperadorProbabilidad, [256](#)
 - OperadorProbabilidad-
 - Homogenea, [258](#)
 - OperadorProbabilidadLineal, [261](#)
 - OperadorProbabilidad-
 - Proporcional, [264](#)
- AsignarProbabilidadMutacion
 - OperadorMutacion, [207](#)
 - OperadorMutacionArreglo, [211](#)
 - OperadorMutacionBoolUniforme, [215](#)
 - OperadorMutacionEntero-
 - Muhlenbein, [219](#)

- OperadorMutacionEnteroNo-Uniforme, [225](#)
- OperadorMutacionEntero-Uniforme, [230](#)
- OperadorMutacionReal-Muhlenbein, [234](#)
- OperadorMutacionRealNo-Uniforme, [240](#)
- OperadorMutacionRealUniforme, [245](#)
- codificacion
 - AlgoritmoGenetico, [58](#)
- Controles
 - AGVentana, [45](#)
- convertArreglo
 - GenArreglo, [89](#)
- copiar
 - Gen, [81](#)
 - GenArreglo, [89](#)
 - GenBool, [99](#)
 - GenEntero, [105](#)
 - GenReal, [113](#)
- crearCopia
 - Gen, [82](#)
 - GenArreglo, [90](#)
 - GenBool, [99](#)
 - GenEntero, [105](#)
 - GenReal, [113](#)
- crearGen
 - GenArreglo, [90](#)
- crearOperadores
 - AlgoritmoGenetico, [59](#)
- CriteriosDeAdaptacion
 - genetico.h, [309](#)
- cruzarGenes
 - OperadorCruce, [140](#)
 - OperadorCruceArreglo, [143](#)
 - OperadorCruceBoolDiscreto, [146](#)
 - OperadorCruceEnteroAritmetico, [149](#)
 - OperadorCruceEnteroBLX, [153](#)
 - OperadorCruceEnteroDiscreto, [156](#)
 - OperadorCruceEnteroHeuristico, [158](#)
 - OperadorCruceEntero-IntermedioExtendido, [161](#)
 - OperadorCruceEnteroLineal, [164](#)
 - OperadorCruceEnteroLineal-BGA, [168](#)
 - OperadorCruceEnteroPlano, [170](#)
 - OperadorCruceRealAritmetico, [172](#)
 - OperadorCruceRealBLX, [177](#)
 - OperadorCruceRealDiscreto, [180](#)
 - OperadorCruceRealHeuristico, [182](#)

OperadorCruceRealIntermedio-
 Extendido, [185](#)
 OperadorCruceRealLineal, [188](#)
 OperadorCruceRealLinealBGA,
 [192](#)
 OperadorCruceRealPlano, [194](#)
 DECLARAR_ALGORITMO
 genetico.h, [307](#)
 DECLARAR_APLICACION
 ventana.h, [316](#)
 DEFINIR_OPERADOR_PAREJAS
 genetico.h, [307](#)
 DEFINIR_OPERADOR_-
 PROBABILIDAD
 genetico.h, [308](#)
 DEFINIR_OPERADOR_-
 REPRODUCCION
 genetico.h, [308](#)
 DEFINIR_OPERADOR_-
 SELECCION
 genetico.h, [308](#)
 definirOperadores
 AlgoritmoGenetico, [60](#)
 Destroy
 Arreglo, [74](#)
 Detach
 Arreglo, [74](#)
 EJECUTAR_EVENTOS
 ventana.h, [317](#)
 ESTADO_CODIFICAR
 genetico.h, [309](#)
 ESTADO_CREAR
 genetico.h, [309](#)
 ESTADO_DECODIFICAR
 genetico.h, [309](#)
 EstadosIndividuo
 genetico.h, [309](#)
 Extremo
 AGVentana, [46](#)
 FIN_DECLARAR_ALGORITMO
 genetico.h, [308](#)
 finalizar
 AlgoritmoGenetico, [60](#)
 OperadorFinalizacion, [196](#)
 OperadorFinalizacionOffline, [199](#)
 OperadorFinalizacionOnline, [203](#)
 finalizarOptimizacion
 AlgoritmoGenetico, [60](#)
 FlushDestroy
 Arreglo, [75](#)
 FlushDetach
 Arreglo, [75](#)
 Gen, [80](#)
 ~Gen, [81](#)
 copiar, [81](#)
 crearCopia, [82](#)
 Gen, [81](#)
 generarAleatorio, [82](#)
 operadorCruceDefecto, [82](#)
 operadorMutacionDefecto, [82](#)

GenArreglo, [84](#)
 GenArreglo, [88](#)
 GenArreglo
 ~GenArreglo, [89](#)
 convertArreglo, [89](#)
 copiar, [89](#)
 crearCopia, [90](#)
 crearGen, [90](#)
 GenArreglo, [88](#)
 generarAleatorio, [90](#)
 getGen, [90](#)
 getGenItems, [91](#)
 getMaxTam, [91](#)
 getMaxVal, [91](#)
 getMinTam, [91](#)
 getMinVal, [91](#)
 getTam, [91](#)
 getVal, [91](#)
 m_pArregloGen, [94](#)
 m_pGenItems, [94](#)
 m_usarVallInicial, [94](#)
 m_vallInicial, [94](#)
 m_valMax, [94](#)
 m_valMin, [94](#)
 operadorCruceDefecto, [92](#)
 operadorMutacionDefecto, [92](#)
 operator=, [92](#), [93](#)
 setTam, [93](#)
 setVal, [93](#)
 genarreglo.h, [288](#)

ADICIONAR_GENARREGLO_-
 BOOL, [291](#)
 ADICIONAR_GENARREGLO_-
 ENTERO, [291](#)
 ADICIONAR_GENARREGLO_-
 REAL, [292](#)
 ArregloBool, [293](#)
 ArregloEntero, [293](#)
 ArregloReal, [293](#)
 GenArregloBool, [293](#)
 GenArregloEntero, [293](#)
 GenArregloReal, [293](#)
 OperadorCruceArregloBool, [293](#)
 OperadorCruceArregloEntero,
 [293](#)
 OperadorCruceArregloReal, [294](#)
 OperadorMutacionArregloBool,
 [294](#)
 OperadorMutacionArregloEnte-
 ro, [294](#)
 OperadorMutacionArregloReal,
 [294](#)
 GenArregloBool
 genarreglo.h, [293](#)
 GenArregloEntero
 genarreglo.h, [293](#)
 GenArregloReal
 genarreglo.h, [293](#)
 GenBool, [96](#)
 GenBool, [98](#)

GenBool
 ~GenBool, 99
 copiar, 99
 crearCopia, 99
 GenBool, 98
 generarAleatorio, 99
 getVal, 99
 m_Valor, 101
 operadorCruceDefecto, 100
 operadorMutacionDefecto, 100
 operator const bool, 100
 operator=, 101
 setVal, 101
 genbool.h, 295
 ADICIONAR_GENBOOL, 295
 GenEntero, 102
 GenEntero, 104, 105
 GenEntero
 ~GenEntero, 105
 copiar, 105
 crearCopia, 105
 GenEntero, 104, 105
 generarAleatorio, 106
 getMax, 106
 getMin, 106
 getVal, 106
 m_Maximo, 108
 m_Minimo, 108
 m_Valor, 108
 operadorCruceDefecto, 106
 operadorMutacionDefecto, 107
 operator const long, 107
 operator=, 107
 setVal, 108
 genentero.h, 297
 ADICIONAR_GENENTERO, 299
 generarAleatorio
 Gen, 82
 GenArreglo, 90
 GenBool, 99
 GenEntero, 106
 GenReal, 114
 Individuo, 122
 genetico.h, 300
 ADAPTACION_-
 PROBMUTACION_-
 EXPONENCIAL, 309
 ADAPTACION_-
 PROBMUTACION_-
 OFFLINE, 309
 ADICIONAR_OPERADOR_-
 ADAPTACION, 306
 ADICIONAR_OPERADOR_-
 CRUCE, 306
 ADICIONAR_OPERADOR_-
 FINALIZACION, 306
 ADICIONAR_OPERADOR_-
 MUTACION, 306
 CriteriosDeAdaptacion, 309
 DECLARAR_ALGORITMO, 307

- DEFINIR_OPERADOR_-
 - PAREJAS, [307](#)
- DEFINIR_OPERADOR_-
 - PROBABILIDAD, [308](#)
- DEFINIR_OPERADOR_-
 - REPRODUCCION, [308](#)
- DEFINIR_OPERADOR_-
 - SELECCION, [308](#)
- ESTADO_CODIFICAR, [309](#)
- ESTADO_CREAR, [309](#)
- ESTADO_DECODIFICAR, [309](#)
- EstadosIndividuo, [309](#)
- FIN_DECLARAR_ALGORITMO, [308](#)
- redondear, [309](#)
- restringir, [309](#)
- GenReal, [110](#)
 - GenReal, [112](#), [113](#)
- GenReal
 - ~GenReal, [113](#)
 - copiar, [113](#)
 - crearCopia, [113](#)
 - generarAleatorio, [114](#)
 - GenReal, [112](#), [113](#)
 - getMax, [114](#)
 - getMin, [114](#)
 - getVal, [114](#)
 - m_Maximo, [116](#)
 - m_Minimo, [116](#)
 - m_Valor, [116](#)
 - operadorCruceDefecto, [114](#)
 - operadorMutacionDefecto, [115](#)
 - operator const double, [115](#)
 - operator=, [115](#)
 - setVal, [116](#)
- genreal.h, [311](#)
 - ADICIONAR_GENREAL, [313](#)
- GetAG
 - Poblacion, [282](#)
- getAlfa
 - OperadorCruceEnteroBLX, [153](#)
 - OperadorCruceRealBLX, [177](#)
- getB
 - OperadorMutacionEnteroNo-Uniforme, [225](#)
 - OperadorMutacionRealNo-Uniforme, [240](#)
- getFactor
 - OperadorMutacionEntero-Muhlenbein, [220](#)
 - OperadorMutacionReal-Muhlenbein, [235](#)
- getGen
 - GenArreglo, [90](#)
 - Individuo, [122](#)
- getGenItems
 - GenArreglo, [91](#)
- getIndividuo
 - Poblacion, [283](#)
- getLambda

- OperadorCruceEnteroAritmetico, 149
 - OperadorCruceRealAritmetico, 173
- getMax
 - GenEntero, 106
 - GenReal, 114
- getMaxTam
 - GenArreglo, 91
- getMaxVal
 - GenArreglo, 91
- getMin
 - GenEntero, 106
 - GenReal, 114
- getMinTam
 - GenArreglo, 91
- getMinVal
 - GenArreglo, 91
- getNmin
 - OperadorProbabilidadLineal, 261
- getObj
 - Arreglo, 75
- getParams
 - OperadorFinalizacionOffline, 199
 - OperadorFinalizacionOnline, 203
- getPareja
 - Individuo, 122
- getProbabilidad
 - Individuo, 122
- getPtr
 - Arreglo, 75
- getSize
 - Arreglo, 76
- getTam
 - GenArreglo, 91
 - Poblacion, 283
- getTamGenoma
 - Individuo, 123
- getVal
 - GenArreglo, 91
 - GenBool, 99
 - GenEntero, 106
 - GenReal, 114
- GrafGrilla
 - AGVentana, 46
- GrafTexto
 - AGVentana, 46
- ID_CONSOLA
 - ventana.h, 317
- ID_GRAFICA
 - ventana.h, 317
- ID_OTRA
 - ventana.h, 317
- Individuo, 118
 - ~Individuo, 121
 - adicionarGen, 121
 - asignarPareja, 121
 - asignarProbabilidad, 122
 - generarAleatorio, 122
 - getGen, 122

- getPareja, [122](#)
- getProbabilidad, [122](#)
- getTamGenoma, [123](#)
- Individuo, [121](#)
- m_Objetivo, [124](#)
- m_objetivoActualizado, [124](#)
- m_pAG, [124](#)
- m_pGenoma, [124](#)
- m_pPareja, [125](#)
- m_Probabilidad, [125](#)
- mutar, [123](#)
- objetivo, [123](#)
- operator=, [123](#)
- remplazarGen, [123](#)
- SetAG, [124](#)
- inicializarApuntadores
 - AlgoritmoGenetico, [60](#)
- inicializarParametros
 - AlgoritmoGenetico, [60](#)
- inicializarVariables
 - AlgoritmoGenetico, [60](#)
- iniciarOptimizacion
 - AlgoritmoGenetico, [61](#)
- Insertar
 - Arreglo, [76](#)
- InsertarIndividuo
 - Poblacion, [283](#)
- IntercambiarPos
 - Arreglo, [76](#)
- iterarOptimizacion
 - AlgoritmoGenetico, [61](#)
- liberarMemoria
 - Arreglo, [77](#)
- m_Alfa
 - OperadorCruceEnteroBLX, [153](#)
 - OperadorCruceRealBLX, [177](#)
- m_alto
 - AGVentana, [48](#)
- m_ancho
 - AGVentana, [48](#)
- m_b
 - OperadorMutacionEnteroNo-Uniforme, [226](#)
 - OperadorMutacionRealNo-Uniforme, [241](#)
- m_capacidad
 - Arreglo, [79](#)
- m_ColBrush
 - AGVentana, [48](#)
- m_ColGraf
 - AGVentana, [48](#)
- m_ColGrilla
 - AGVentana, [48](#)
- m_Columna
 - AGVentana, [48](#)
- m_congelar
 - AGFrame, [38](#)
- m_contador
 - OperadorFinalizacionOffline, [200](#)
 - OperadorFinalizacionOnline, [204](#)

m_Desviacion	AlgoritmoGenetico, 64
m_dib	AGVentana, 48
m_dibCopia	AGVentana, 48
m_escalon	OperadorAdaptacionProb- Mutacion, 137
m_Factor	OperadorMutacionEntero- Muhlenbein, 221 OperadorMutacionReal- Muhlenbein, 236
m_factorVariacion	OperadorAdaptacionProb- Mutacion, 137 OperadorFinalizacionOffline, 200 OperadorFinalizacionOnline, 204
m_Generacion	AlgoritmoGenetico, 64
m_GeneracionDelMejorEnLaHistoria	AlgoritmoGenetico, 64
m_GeneracionMaxima	AlgoritmoGenetico, 65
m_grafica	AGFrame, 38
m_IndicadorArchivo	AlgoritmoGenetico, 65
m_IndicadorInicializarPoblacionAleatoria	AlgoritmoGenetico, 65
m_IndicadorMaximizar	AlgoritmoGenetico, 65
m_IndicadorMostrar	AlgoritmoGenetico, 65
m_IndicadorMostrarDesviacion	AlgoritmoGenetico, 65
m_IndicadorMostrarGeneracionMejorHistorico	AlgoritmoGenetico, 65
m_IndicadorMostrarMedia	AlgoritmoGenetico, 65
m_IndicadorMostrarMejorEnGeneracion	AlgoritmoGenetico, 65
m_IndicadorMostrarMejorEnHistoria	AlgoritmoGenetico, 66
m_IndicadorMostrarOffLine	AlgoritmoGenetico, 66
m_IndicadorMostrarOnLine	AlgoritmoGenetico, 66
m_IndicadorMostrarPeorEnGeneracion	AlgoritmoGenetico, 66
m_IndicadorUsarAdaptacion	AlgoritmoGenetico, 66
m_Ini	AGVentana, 48
m_IntervaloSalvar	AlgoritmoGenetico, 66
m_items	Arreglo, 79
m_Lambda	

OperadorCruceEnteroAritmetico, [149](#)
 OperadorCruceRealAritmetico, [173](#)
 m_max
 AGVentana, [48](#)
 m_maxCont
 OperadorAdaptacionProb-
 Mutacion, [137](#)
 OperadorFinalizacionOffline, [200](#)
 OperadorFinalizacionOnline, [204](#)
 m_Maximo
 GenEntero, [108](#)
 GenReal, [116](#)
 m_maxProb
 OperadorAdaptacionProb-
 Mutacion, [137](#)
 m_Media
 AlgoritmoGenetico, [66](#)
 m_MedidaOffLine
 AlgoritmoGenetico, [66](#)
 m_MedidaOffLineAnterior
 AlgoritmoGenetico, [66](#)
 m_MedidaOnLine
 AlgoritmoGenetico, [67](#)
 m_MedidaOnLineAnterior
 AlgoritmoGenetico, [67](#)
 m_min
 AGVentana, [48](#)
 m_Minimo
 GenEntero, [108](#)
 GenReal, [116](#)
 m_NameArch
 AGVentana, [48](#)
 m_nIndivFin
 OperadorAdaptacionNum-
 Individuos, [132](#)
 m_nIndivInicio
 OperadorAdaptacionNum-
 Individuos, [132](#)
 m_nmin
 OperadorProbabilidadLineal, [261](#)
 m_NomArch
 AGFrame, [39](#)
 m_NombreArchivo
 AlgoritmoGenetico, [67](#)
 m_NumGen
 AGVentana, [49](#)
 m_Objetivo
 Individuo, [124](#)
 m_objetivoActualizado
 Individuo, [124](#)
 m_pAG
 Individuo, [124](#)
 OperadorCruceEnteroLineal-
 BGA, [168](#)
 OperadorCruceRealLinealBGA,
 [192](#)
 OperadorMutacionEnteroNo-
 Uniforme, [226](#)

OperadorMutacionRealNo-	AlgoritmoGenetico, 67
Uniforme, 241	
Poblacion, 285	m_pMejorEnLaHistoria
	AlgoritmoGenetico, 68
m_pArregloGen	m_pMenuBar
GenArreglo, 94	AGFrame, 39
m_pBitmap	m_pMenuGrafica
AGVentana, 49	AGFrame, 39
m_pCongelarMenu	m_pModelo
AGFrame, 39	AlgoritmoGenetico, 68
m_pData	m_pNotebook
Arreglo, 79	AGFrame, 39
m_pFileMenu	m_pOperadorCruceGenes
AGFrame, 39	OperadorCruceArreglo, 144
m_pGeneracion	m_pOperadorCruceTamanos
Poblacion, 285	OperadorCruceArreglo, 144
m_pGenItems	m_pOperadorMutacionGenes
GenArreglo, 94	OperadorMutacionArreglo, 212
m_pGenoma	m_pOpParejas
Individuo, 124	AlgoritmoGenetico, 68
m_pInfoMenu	m_pOpProbabilidad
AGFrame, 39	AlgoritmoGenetico, 68
m_pListaOperadorAdaptacion	m_pOpReproduccion
AlgoritmoGenetico, 67	AlgoritmoGenetico, 68
m_pListaOperadorCruce	m_pOpSeleccion
AlgoritmoGenetico, 67	AlgoritmoGenetico, 68
m_pListaOperadorFinalizacion	m_pPagina
AlgoritmoGenetico, 67	AGFrame, 39
m_pListaOperadorMutacion	m_pPanel
AlgoritmoGenetico, 67	AGFrame, 39
m_pMejorEnEstaGeneracion	m_pPareja

Individuo, [125](#)
 m_pPeorEnEstaGeneracion
 AlgoritmoGenetico, [68](#)
 m_pPoblacionActual
 AlgoritmoGenetico, [68](#)
 m_Probabilidad
 Individuo, [125](#)
 m_ProbabilidadMutacion
 OperadorMutacion, [208](#)
 OperadorMutacionArreglo, [212](#)
 OperadorMutacionBoolUniforme,
 [216](#)
 OperadorMutacionEntero-
 Muhlenbein, [221](#)
 OperadorMutacionEnteroNo-
 Uniforme, [226](#)
 OperadorMutacionEntero-
 Uniforme, [231](#)
 OperadorMutacionReal-
 Muhlenbein, [236](#)
 OperadorMutacionRealNo-
 Uniforme, [241](#)
 OperadorMutacionRealUniforme,
 [246](#)
 m_pSizer
 AGVentana, [49](#)
 m_pSizerFrame
 AGFrame, [39](#)
 m_pSizerNB
 AGFrame, [39](#)

m_pTextarch
 AGVentana, [49](#)
 m_pTextCtrl
 AGFrame, [40](#)
 m_pTxtConsola
 AGVentana, [49](#)
 m_pValores
 AGVentana, [49](#)
 m_T
 OperadorAdaptacionProb-
 Mutacion, [137](#)
 m_tam
 AGVentana, [49](#)
 m_TamanoPoblacion
 AlgoritmoGenetico, [68](#)
 m_tipoAdaptacion
 OperadorAdaptacionProb-
 Mutacion, [137](#)
 m_Titulo
 AGVentana, [49](#)
 m_usarVallInicial
 GenArreglo, [94](#)
 m_vallInicial
 GenArreglo, [94](#)
 m_valMax
 GenArreglo, [94](#)
 m_valMin
 GenArreglo, [94](#)
 m_Valor
 GenBool, [101](#)

- GenEntero, [108](#)
- GenReal, [116](#)
- MENU_CONGELAR
 - AGFrame, [36](#)
- MENU_FILE_QUIT
 - AGFrame, [35](#)
- MENU_FILE_SAVE
 - AGFrame, [35](#)
- MENU_GRAFICA_FONDO
 - AGFrame, [35](#)
- MENU_GRAFICA_GRAFCAR
 - AGFrame, [36](#)
- MENU_GRAFICA_GRILLA
 - AGFrame, [35](#)
- MENU_GRAFICA_LINEA
 - AGFrame, [35](#)
- MENU_INFO_ABOUT
 - AGFrame, [35](#)
- menus
 - AGFrame, [35](#)
- mostrarMedidas
 - AlgoritmoGenetico, [61](#)
- mutar
 - AlgoritmoGenetico, [62](#)
 - Individuo, [123](#)
 - OperadorMutacion, [207](#)
 - OperadorMutacionArreglo, [211](#)
 - OperadorMutacionBoolUniforme, [215](#)
 - OperadorMutacionEntero-

- Muhlenbein, [220](#)
- OperadorMutacionEnteroNo-Uniforme, [225](#)
- OperadorMutacionEntero-Uniforme, [230](#)
- OperadorMutacionReal-Muhlenbein, [235](#)
- OperadorMutacionRealNo-Uniforme, [240](#)
- OperadorMutacionRealUniforme, [245](#)
- Poblacion, [283](#)
- mutarGen
 - OperadorMutacion, [208](#)
 - OperadorMutacionArreglo, [212](#)
 - OperadorMutacionBoolUniforme, [215](#)
 - OperadorMutacionEntero-Muhlenbein, [220](#)
 - OperadorMutacionEnteroNo-Uniforme, [225](#)
 - OperadorMutacionEntero-Uniforme, [230](#)
 - OperadorMutacionReal-Muhlenbein, [235](#)
 - OperadorMutacionRealNo-Uniforme, [240](#)
 - OperadorMutacionRealUniforme, [245](#)

NOTEBOOK

AGFrame, [36](#)
 objetivo
 AlgoritmoGenetico, [62](#)
 Individuo, [123](#)
 ObtenerExtremos
 AGVentana, [47](#)
 ObtenerFrame
 AGVentana, [47](#)
 ObtenerPagina
 AGFrame, [36](#)
 ObtenerProbabilidadMutacion
 OperadorMutacion, [208](#)
 OperadorMutacionArreglo, [212](#)
 OperadorMutacionBoolUniforme,
 [215](#)
 OperadorMutacionEntero-
 Muhlenbein, [220](#)
 OperadorMutacionEnteroNo-
 Uniforme, [226](#)
 OperadorMutacionEntero-
 Uniforme, [230](#)
 OperadorMutacionReal-
 Muhlenbein, [235](#)
 OperadorMutacionRealNo-
 Uniforme, [241](#)
 OperadorMutacionRealUniforme,
 [245](#)
 ObtenerValoresGrafica
 AGVentana, [47](#)
 ObtenerY
 AGVentana, [47](#)
 OnClose
 AGFrame, [37](#)
 OnCongelar
 AGFrame, [37](#)
 OnMaxLen
 AGVentana, [47](#)
 OnMenuFileQuit
 AGFrame, [37](#)
 OnMenuFileSave
 AGFrame, [37](#)
 OnMenuFondo
 AGFrame, [37](#)
 OnMenuGrafica
 AGFrame, [37](#)
 OnMenuGrilla
 AGFrame, [37](#)
 OnMenuInfoAbout
 AGFrame, [38](#)
 OnMenuLinea
 AGFrame, [38](#)
 OnPaint
 AGVentana, [47](#)
 OperadorAdaptacion, [126](#)
 OperadorAdaptacion, [127](#)
 OperadorAdaptacion
 ~OperadorAdaptacion, [127](#)
 adaptacion, [127](#)
 OperadorAdaptacion, [127](#)
 OperadorAdaptacionElitismo, [128](#)

OperadorAdaptacionElitismo,	m_maxCont, 137
129	m_maxProb, 137
OperadorAdaptacionElitismo	m_T, 137
~OperadorAdaptacionElitismo,	m_tipoAdaptacion, 137
129	OperadorAdaptacionProbMuta-
adaptacion, 129	cion, 135
OperadorAdaptacionElitismo,	setParamsExponencial, 136
129	setParamsOffline, 136
OperadorAdaptacionNumIndividuos,	OperadorCruce, 139
130	OperadorCruce, 140
OperadorAdaptacionNum-	OperadorCruce
Individuos, 131	~OperadorCruce, 140
OperadorAdaptacionNumIndividuos	cruzarGenes, 140
~OperadorAdaptacionNumIndividuos,	OperadorCruce, 140
131	OperadorCruceArreglo, 142
adaptacion, 132	OperadorCruceArreglo, 143
m_nIndivFin, 132	OperadorCruceArreglo
m_nIndivInicio, 132	~OperadorCruceArreglo, 143
OperadorAdaptacionNumIndivi-	cruzarGenes, 143
duos, 131	m_pOperadorCruceGenes, 144
OperadorAdaptacionProbMutacion,	m_pOperadorCruceTamanos,
133	144
OperadorAdaptacionProb-	OperadorCruceArreglo, 143
Mutacion, 135	OperadorCruceArregloBool
OperadorAdaptacionProbMutacion	genarreglo.h, 293
~OperadorAdaptacionProbMutacion,	OperadorCruceArregloEntero
136	genarreglo.h, 293
adaptacion, 136	OperadorCruceArregloReal
m_escalon, 137	genarreglo.h, 294
m_factorVariacion, 137	OperadorCruceBoolDiscreto, 145

OperadorCruceBoolDiscreto,	~OperadorCruceEnteroBLX,
146	152
OperadorCruceBoolDiscreto	cruzarGenes, 153
~OperadorCruceBoolDiscreto,	getAlfa, 153
146	m_Alfa, 153
cruzarGenes, 146	OperadorCruceEnteroBLX, 152
OperadorCruceBoolDiscreto,	setAlfa, 153
146	OperadorCruceEnteroDiscreto, 155
operadorCruceDefecto	OperadorCruceEnteroDiscreto,
Gen, 82	156
GenArreglo, 92	OperadorCruceEnteroDiscreto
GenBool, 100	~OperadorCruceEnteroDiscreto,
GenEntero, 106	156
GenReal, 114	cruzarGenes, 156
OperadorCruceEnteroAritmetico,	OperadorCruceEnteroDiscreto,
147	156
OperadorCruceEnteroAritmetico,	OperadorCruceEnteroHeuristico,
148	157
OperadorCruceEnteroAritmetico	OperadorCruceEnteroHeuristico,
~OperadorCruceEnteroAritmetico,	158
148	OperadorCruceEnteroHeuristico
cruzarGenes, 149	~OperadorCruceEnteroHeuristico,
getLambda, 149	158
m_Lambda, 149	cruzarGenes, 158
OperadorCruceEnteroAritmetico,	OperadorCruceEnteroHeuristico,
148	158
setLambda, 149	OperadorCruceEnteroIntermedioExtendido,
OperadorCruceEnteroBLX, 151	160
OperadorCruceEnteroBLX, 152	OperadorCruceEntero-
OperadorCruceEnteroBLX	IntermedioExtendido, 161

OperadorCruceEnteroIntermedio- Extendido	~OperadorCruceEnteroPlano, 170
~OperadorCruceEnteroIntermedioExtendido, 161	cruzarGenes, 170
cruzarGenes, 161	OperadorCruceEnteroPlano, 170
OperadorCruceEnteroInterme- dioExtendido, 161	OperadorCruceRealAritmetico, 171
OperadorCruceEnteroLineal, 163	OperadorCruceRealAritmetico, 172
OperadorCruceEnteroLineal, 164	OperadorCruceRealAritmetico ~OperadorCruceRealAritmetico, 172
OperadorCruceEnteroLineal ~OperadorCruceEnteroLineal, 164	cruzarGenes, 172
cruzarGenes, 164	getLambda, 173
OperadorCruceEnteroLineal, 164	m_Lambda, 173
OperadorCruceEnteroLinealBGA, 166	OperadorCruceRealAritmetico, 172
OperadorCruceEnteroLineal- BGA, 167	setLambda, 173
OperadorCruceEnteroLinealBGA ~OperadorCruceEnteroLinealBGA, 167	OperadorCruceRealBLX, 175
cruzarGenes, 168	OperadorCruceRealBLX, 176
m_pAG, 168	OperadorCruceRealBLX ~OperadorCruceRealBLX, 176
OperadorCruceEnteroLinealB- GA, 167	cruzarGenes, 177
OperadorCruceEnteroPlano, 169	getAlfa, 177
OperadorCruceEnteroPlano, 170	m_Alfa, 177
OperadorCruceEnteroPlano	OperadorCruceRealBLX, 176
	setAlfa, 177
	OperadorCruceRealDiscreto, 179
	OperadorCruceRealDiscreto, 180
	OperadorCruceRealDiscreto ~OperadorCruceRealDiscreto,

[180](#)
 cruzarGenes, [180](#)
 OperadorCruceRealDiscreto,
[180](#)
 OperadorCruceRealHeuristico, [181](#)
 OperadorCruceRealHeuristico,
[182](#)
 OperadorCruceRealHeuristico
 ~OperadorCruceRealHeuristico,
[182](#)
 cruzarGenes, [182](#)
 OperadorCruceRealHeuristico,
[182](#)
 OperadorCruceRealIntermedioExtendido,
[184](#)
 OperadorCruceRealIntermedio-
 Extendido, [185](#)
 OperadorCruceRealIntermedio-
 Extendido
 ~OperadorCruceRealIntermedioExtendido, [185](#)
 cruzarGenes, [185](#)
 OperadorCruceRealInterme-
 dioExtendido, [185](#)
 OperadorCruceRealLineal, [187](#)
 OperadorCruceRealLineal, [188](#)
 OperadorCruceRealLineal
 ~OperadorCruceRealLineal, [188](#)
 cruzarGenes, [188](#)
 OperadorCruceRealLineal, [188](#)
 OperadorCruceRealLinealBGA, [190](#)
 OperadorCruceRealLinealBGA,
[191](#)
 OperadorCruceRealLinealBGA
 ~OperadorCruceRealLinealBGA,
[191](#)
 cruzarGenes, [192](#)
 m_pAG, [192](#)
 OperadorCruceRealLinealBGA,
[191](#)
 OperadorCruceRealPlano, [193](#)
 OperadorCruceRealPlano, [194](#)
 OperadorCruceRealPlano
 ~OperadorCruceRealPlano, [194](#)
 cruzarGenes, [194](#)
 OperadorCruceRealPlano, [194](#)
 OperadorFinalizacion, [195](#)
 OperadorFinalizacion, [195](#)
 OperadorFinalizacion
 finalizar, [196](#)
 OperadorFinalizacion, [195](#)
 OperadorFinalizacionOffline, [197](#)
 OperadorFinalizacionOffline, [198](#)
 OperadorFinalizacionOffline
 ~OperadorFinalizacionOffline,
[199](#)
 finalizar, [199](#)
 getParams, [199](#)
 m_contador, [200](#)

- m_factorVariacion, [200](#)
- m_maxCont, [200](#)
- OperadorFinalizacionOffline, [198](#)
- setParams, [199](#)
- OperadorFinalizacionOnline, [201](#)
 - OperadorFinalizacionOnline, [202](#)
- OperadorFinalizacionOnline
 - ~OperadorFinalizacionOnline, [203](#)
 - finalizar, [203](#)
 - getParams, [203](#)
 - m_contador, [204](#)
 - m_factorVariacion, [204](#)
 - m_maxCont, [204](#)
 - OperadorFinalizacionOnline, [202](#)
 - setParams, [203](#)
- OperadorMutacion, [205](#)
 - OperadorMutacion, [207](#)
- OperadorMutacion
 - ~OperadorMutacion, [207](#)
 - AsignarProbabilidadMutacion, [207](#)
 - m_ProbabilidadMutacion, [208](#)
 - mutar, [207](#)
 - mutarGen, [208](#)
 - ObtenerProbabilidadMutacion, [208](#)
 - OperadorMutacion, [207](#)
- OperadorMutacionArreglo, [209](#)
 - OperadorMutacionArreglo, [211](#)
- OperadorMutacionArreglo
 - ~OperadorMutacionArreglo, [211](#)
 - AsignarProbabilidadMutacion, [211](#)
 - m_pOperadorMutacionGenes, [212](#)
 - m_ProbabilidadMutacion, [212](#)
 - mutar, [211](#)
 - mutarGen, [212](#)
 - ObtenerProbabilidadMutacion, [212](#)
 - OperadorMutacionArreglo, [211](#)
- OperadorMutacionArregloBool
 - genarreglo.h, [294](#)
- OperadorMutacionArregloEntero
 - genarreglo.h, [294](#)
- OperadorMutacionArregloReal
 - genarreglo.h, [294](#)
- OperadorMutacionBoolUniforme, [213](#)
 - OperadorMutacionBoolUniforme, [214](#)
- OperadorMutacionBoolUniforme
 - ~OperadorMutacionBoolUniforme, [214](#)
 - AsignarProbabilidadMutacion, [215](#)
 - m_ProbabilidadMutacion, [216](#)
 - mutar, [215](#)
 - mutarGen, [215](#)

ObtenerProbabilidadMutacion,	OperadorMutacionEnteroNoUniforme,
215	222
OperadorMutacionBoolUniforme,	OperadorMutacionEnteroNo-
214	Uniforme, 224
operadorMutacionDefecto	OperadorMutacionEnteroNo-
Gen, 82	Uniforme
GenArreglo, 92	~OperadorMutacionEnteroNoUniforme,
GenBool, 100	224
GenEntero, 107	AsignarProbabilidadMutacion,
GenReal, 115	225
OperadorMutacionEnteroMuhlenbein,	getB, 225
217	m_b, 226
OperadorMutacionEntero-	m_pAG, 226
Muhlenbein, 219	m_ProbabilidadMutacion, 226
OperadorMutacionEntero-	mutar, 225
Muhlenbein	mutarGen, 225
~OperadorMutacionEnteroMuhlenbein,	ObtenerProbabilidadMutacion,
219	226
AsignarProbabilidadMutacion,	OperadorMutacionEnteroNoUni-
219	forme, 224
getFactor, 220	setB, 226
m_Factor, 221	OperadorMutacionEnteroUniforme,
m_ProbabilidadMutacion, 221	228
mutar, 220	OperadorMutacionEntero-
mutarGen, 220	Uniforme, 229
ObtenerProbabilidadMutacion,	OperadorMutacionEnteroUniforme
220	~OperadorMutacionEnteroUniforme,
OperadorMutacionEnteroMuh-	229
lenbein, 219	AsignarProbabilidadMutacion,
setFactor, 220	230

m_ProbabilidadMutacion, 231	OperadorMutacionRealNoUniforme
mutar, 230	~OperadorMutacionRealNoUniforme, 239
mutarGen, 230	AsignarProbabilidadMutacion, 240
ObtenerProbabilidadMutacion, 230	getB, 240
OperadorMutacionEnteroUniforme, 229	m_b, 241
OperadorMutacionRealMuhlenbein, 232	m_pAG, 241
OperadorMutacionRealMuhlenbein, 234	m_ProbabilidadMutacion, 241
AsignarProbabilidadMutacion, 234	mutar, 240
getFactor, 235	mutarGen, 240
m_Factor, 236	ObtenerProbabilidadMutacion, 241
m_ProbabilidadMutacion, 236	OperadorMutacionRealNoUniforme, 239
mutar, 235	setB, 241
mutarGen, 235	OperadorMutacionRealUniforme, 243
ObtenerProbabilidadMutacion, 235	OperadorMutacionRealUniforme, 244
OperadorMutacionRealMuhlenbein, 234	OperadorMutacionRealUniforme
setFactor, 236	~OperadorMutacionRealUniforme, 244
OperadorMutacionRealNoUniforme, 237	AsignarProbabilidadMutacion, 245
OperadorMutacionRealNoUniforme, 239	m_ProbabilidadMutacion, 246
	mutar, 245
	mutarGen, 245
	ObtenerProbabilidadMutacion, 245

OperadorMutacionRealUniforme, [244](#)
 OperadorParejas, [247](#)
 OperadorParejas, [248](#)
 OperadorParejas
 ~OperadorParejas, [248](#)
 asignarParejas, [248](#)
 OperadorParejas, [248](#)
 OperadorParejasAdyacentes, [249](#)
 OperadorParejasAdyacentes, [250](#)
 OperadorParejasAdyacentes
 ~OperadorParejasAdyacentes, [250](#)
 asignarParejas, [250](#)
 OperadorParejasAdyacentes, [250](#)
 OperadorParejasAleatorias, [251](#)
 OperadorParejasAleatorias, [252](#)
 OperadorParejasAleatorias
 ~OperadorParejasAleatorias, [252](#)
 asignarParejas, [252](#)
 OperadorParejasAleatorias, [252](#)
 OperadorParejasExtremos, [253](#)
 OperadorParejasExtremos, [254](#)
 OperadorParejasExtremos
 ~OperadorParejasExtremos, [254](#)
 asignarParejas, [254](#)
 OperadorParejasExtremos, [254](#)
 OperadorProbabilidad, [255](#)
 OperadorProbabilidad, [256](#)
 OperadorProbabilidad
 ~OperadorProbabilidad, [256](#)
 asignarProbabilidad, [256](#)
 OperadorProbabilidad, [256](#)
 OperadorProbabilidadHomogenea, [257](#)
 OperadorProbabilidad-Homogenea, [258](#)
 OperadorProbabilidadHomogenea
 ~OperadorProbabilidadHomogenea, [258](#)
 asignarProbabilidad, [258](#)
 OperadorProbabilidadHomogenea, [258](#)
 OperadorProbabilidadLineal, [259](#)
 OperadorProbabilidadLineal, [260](#)
 OperadorProbabilidadLineal
 ~OperadorProbabilidadLineal, [260](#)
 asignarProbabilidad, [261](#)
 getNmin, [261](#)
 m_nmin, [261](#)
 OperadorProbabilidadLineal, [260](#)
 setNmin, [261](#)
 OperadorProbabilidadProporcional, [263](#)
 OperadorProbabilidad-

Proporcional, [264](#)
 OperadorProbabilidadProporcional
 ~OperadorProbabilidadProporcional,
 [264](#)
 asignarProbabilidad, [264](#)
 OperadorProbabilidadProporcional,
 [264](#)
 OperadorReproduccion, [266](#)
 OperadorReproduccion, [267](#)
 OperadorReproduccion
 ~OperadorReproduccion, [267](#)
 OperadorReproduccion, [267](#)
 reproducir, [267](#)
 OperadorReproduccionCruceSimple,
 [268](#)
 OperadorReproduccionCruceSimple,
 [269](#)
 OperadorReproduccionCruceSimple
 ~OperadorReproduccionCruceSimple,
 [269](#)
 OperadorReproduccionCruceSimple,
 [269](#)
 reproducir, [269](#)
 OperadorReproduccionDosPadresDosHijos,
 [270](#)
 OperadorReproduccionDosPadresDosHijos,
 [271](#)
 OperadorReproduccionDosPadresDosHijos
 ~OperadorReproduccionDosPadresDosHijos,
 [271](#)
 OperadorReproduccionDosPadresDosHijos,
 [271](#)
 reproducir, [271](#)
 OperadorReproduccionMejoresEntrePadresEHijos,
 [272](#)
 OperadorReproduccionMejoresEntrePadresEHijos,
 [273](#)
 OperadorReproduccionMejoresEntrePadresEHijos
 ~OperadorReproduccionMejoresEntrePadresEHijos,
 [273](#)
 OperadorReproduccionMejoresEntrePadresEHijos,
 [273](#)
 reproducir, [273](#)
 OperadorReproduccionMejorPadreMejorHijo,
 [274](#)
 OperadorReproduccionMejorPadreMejorHijo,
 [275](#)
 OperadorReproduccionMejorPadreMejorHijo
 ~OperadorReproduccionMejorPadreMejorHijo,
 [275](#)
 OperadorReproduccionMejorPadreMejorHijo,
 [275](#)
 reproducir, [275](#)
 OperadorSeleccion, [276](#)
 OperadorSeleccion, [277](#)
 OperadorSeleccion
 OperadorSeleccion, [277](#)

OperadorSeleccion, 277	AlgoritmoGenetico, 63
seleccionar, 277	ordenar
OperadorSeleccionEstocasticaRemplazo, 278	Poblacion, 284
OperadorSeleccionEstocastica-Remplazo, 279	OTRO
OperadorSeleccionEstocastica-Remplazo	AGVentana, 45
~OperadorSeleccionEstocasticaRemplazo, 279	Paginas
OperadorSeleccionEstocastica-Remplazo, 279	ventana.h, 317
seleccionar, 279	Poblacion, 280
operator const bool	~Poblacion, 282
GenBool, 100	GetAG, 282
operator const double	getIndividuo, 283
GenReal, 115	getTam, 283
operator const long	InsertarIndividuo, 283
GenEntero, 107	m_pAG, 285
operator=	m_pGeneracion, 285
Arreglo, 77	mutar, 283
GenArreglo, 92 , 93	operator=, 284
GenBool, 101	ordenar, 284
GenEntero, 107	Poblacion, 282
GenReal, 115	reemplazarIndividuo, 284
Individuo, 123	SetAG, 285
Poblacion, 284	setTam, 285
operator[]	redondear
Arreglo, 77	genetico.h, 309
optimizar	reemplazar
	Arreglo, 78
	reemplazarGen
	Individuo, 123
	reemplazarIndividuo
	Poblacion, 284

- reproducir
 - AlgoritmoGenetico, [63](#)
 - OperadorReproduccion, [267](#)
 - OperadorReproduccionCruce-
 - Simple, [269](#)
 - OperadorReproduccionDos-
 - PadresDosHijos, [271](#)
 - OperadorReproduccionMejores-
 - EntrePadresEHijos, [273](#)
 - OperadorReproduccionMejor-
 - PadreMejorHijo, [275](#)
- restringir
 - genetico.h, [309](#)
- salvar
 - AlgoritmoGenetico, [63](#)
- SelColor
 - AGFrame, [38](#)
- seleccionar
 - AlgoritmoGenetico, [64](#)
 - OperadorSeleccion, [277](#)
 - OperadorSeleccionEstocastica-
 - Remplazo, [279](#)
- SetAG
 - Individuo, [124](#)
 - Poblacion, [285](#)
- setAlfa
 - OperadorCruceEnteroBLX, [153](#)
 - OperadorCruceRealBLX, [177](#)
- setB
 - OperadorMutacionEnteroNo-
 - Uniforme, [226](#)
 - OperadorMutacionRealNo-
 - Uniforme, [241](#)
- setFactor
 - OperadorMutacionEntero-
 - Muhlenbein, [220](#)
 - OperadorMutacionReal-
 - Muhlenbein, [236](#)
- setLambda
 - OperadorCruceEnteroAritmetico,
 - [149](#)
 - OperadorCruceRealAritmetico,
 - [173](#)
- setNmin
 - OperadorProbabilidadLineal, [261](#)
- SetPagina
 - AGFrame, [38](#)
- setParams
 - OperadorFinalizacionOffline, [199](#)
 - OperadorFinalizacionOnline, [203](#)
- setParamsExponencial
 - OperadorAdaptacionProb-
 - Mutacion, [136](#)
- setParamsOffline
 - OperadorAdaptacionProb-
 - Mutacion, [136](#)
- setTam
 - GenArreglo, [93](#)
 - Poblacion, [285](#)
- setVal

GenArreglo, [93](#)

GenBool, [101](#)

GenEntero, [108](#)

GenReal, [116](#)

TEXT_CONSOLA

AGVentana, [45](#)

Truncar

Arreglo, [78](#)

UNGenetico.h, [314](#)

ventana.h, [315](#)

DECLARAR_APLICACION, [316](#)

EJECUTAR_EVENTOS, [317](#)

ID_CONSOLA, [317](#)

ID_GRAFICA, [317](#)

ID_OTRA, [317](#)

Paginas, [317](#)