Population Options

Population options let you specify the parameters of the population that the genetic algorithm uses.

Population type (`PopulationType`) specifies the type of input to the fitness function. Types and their restrictions are:

- `Double vector` (`'doubleVector'`) — Use this option if the individuals in the population have type `double`. Use this option for mixed integer programming. This is the default.
- `Bit string` (`'bitstring'`) — Use this option if the individuals in the population have components that are `0` or `1`.

  > **Caution** The individuals in a `Bit string` population are vectors of type `double`, not strings.

  For **Creation function** (`CreationFcn`) and **Mutation function** (`MutationFcn`), use `Uniform` (`@gacreationuniform` and `@mutationuniform`) or `Custom`. For **Crossover function** (`CrossoverFcn`), use `Scattered` (`@crossoverscattered`), `Single point` (`@crossoversinglepoint`), `Two point` (`@crossovertwopoint`), or `Custom`. You cannot use a **Hybrid function**, and `ga` ignores all constraints, including bounds, linear constraints, and nonlinear constraints.
- `Custom` — For **Crossover function** and **Mutation function**, use `Custom`. For **Creation function**, either use `Custom`, or provide an **Initial population**. You cannot use a **Hybrid function**, and `ga` ignores all constraints, including bounds, linear constraints, and nonlinear constraints.

Population size (`PopulationSize`) specifies how many individuals there are in each generation. With a large population size, the genetic algorithm searches the solution space more thoroughly, thereby reducing the chance that the algorithm returns a local minimum that is not a global minimum. However, a large population size also causes the algorithm to run more slowly.

If you set **Population size** to a vector, the genetic algorithm creates multiple subpopulations, the number of which is the length of the vector. The size of each subpopulation is the corresponding entry of the vector. See Migration Options.

Creation function (`CreationFcn`) specifies the function that creates the initial population for `ga`. Do not specify a creation function with integer problems because `ga` overrides any choice you make. Choose from:

- `[]` uses the default creation function for your problem.
- `Uniform` (`@gacreationuniform`) creates a random initial population with a uniform distribution. This is the default when there are no linear constraints, or when there are integer constraints. The uniform distribution is in the initial population range (`PopInitRange`). The default values for `PopInitRange` are `[-10;10]` for every component, or `[-9999;10001]` when there are integer constraints. These bounds are shifted and scaled to match any existing bounds `lb` and `ub`.

  > **Caution** Do not use `@gacreationuniform` when you have linear constraints. Otherwise, your population might not satisfy the linear constraints.

- `Feasible population` (`@gacreationlinearfeasible`), the default when there are linear constraints and no integer constraints, creates a random initial population that satisfies all bounds and linear constraints. If there are linear constraints, `Feasible population` creates many individuals on the boundaries of the constraint region, and creates a well-dispersed population. `Feasible population` ignores **Initial range** (`PopInitRange`).

  `gacreationlinearfeasible` calls `linprog` to create a feasible population with respect to bounds and linear constraints.

  For an example showing its behavior, see Linearly Constrained Population and Custom Plot Function.

- `Nonlinear Feasible population` (`@gacreationnonlinearfeasible`) is the default creation function for the `'penalty'` nonlinear constraint algorithm. For details, see Constraint Parameters.

- `Custom` lets you write your own creation function, which must generate data of the type that you specify in **Population type**. To specify the creation function if you are using the Optimization app,

  - Set **Creation function** to `Custom`.
  - Set **Function name** to `@myfun`, where `myfun` is the name of your function.

  If you are using `ga`, set

  ```
  options = gaoptimset('CreationFcn', @myfun);
  ```

  Your creation function must have the following calling syntax.

  ```
  function Population = myfun(GenomeLength, FitnessFcn, options)
  ```

  The input arguments to the function are:

  - `Genomelength` — Number of independent variables for the fitness function
  - `FitnessFcn` — Fitness function
  - `options` — Options structure

  The function returns `Population`, the initial population for the genetic algorithm.

  Passing Extra Parameters in the Optimization Toolbox documentation explains how to provide additional parameters to the function.

  > **Caution** When you have bounds or linear constraints, ensure that your creation function creates individuals that satisfy these constraints. Otherwise, your population might not satisfy the constraints.

Initial population (`InitialPopulation`) specifies an initial population for the genetic algorithm. The default value is `[]`, in which case `ga` uses the default **Creation function** to create an initial population. If you enter a nonempty array in the **Initial population** field, the array must have no more than **Population size** rows, and exactly **Number of variables** columns. In this case, the genetic algorithm calls a **Creation function** to generate the remaining individuals, if required.

Initial scores (`InitialScores`) specifies initial scores for the initial population. The initial scores can also be partial. Do not specify initial scores with integer problems because `ga` overrides any choice you make.

Initial range (`PopInitRange`) specifies the range of the vectors in the initial population that is generated by the `gacreationuniform` creation function. You can set **Initial range** to be a matrix with two rows and **Number of variables** columns, each column of which has the form `[lb;ub]`, where `lb` is the lower bound and `ub` is the upper bound for the entries in that coordinate. If you specify **Initial range** to be a 2-by-1 vector, each entry is expanded to a constant row of length **Number of variables**. If you do not specify an **Initial range**, the default is `[-10;10]` (`[-1e4+1;1e4+1]` for integer-constrained problems), modified to match any existing bounds.

See Setting the Initial Range for an example.