Search MATLAB Documentation Search Documentation Search

- R2012b
- Global Optimization Toolbox
  ...

ga

Find minimum of function using genetic algorithm

Syntax

```
x = ga(fitnessfcn,nvars)
x = ga(fitnessfcn,nvars,A,b)
x = ga(fitnessfcn,nvars,A,b,Aeq,beq)
x = ga(fitnessfcn,nvars,A,b,Aeq,beq,LB,UB)
x = ga(fitnessfcn,nvars,A,b,Aeq,beq,LB,UB,nonlcon)
x = ga(fitnessfcn,nvars,A,b,Aeq,beq,LB,UB,nonlcon,options)
x = ga(fitnessfcn,nvars,A,b,[],[],LB,UB,nonlcon,IntCon)
x = ga(fitnessfcn,nvars,A,b,[],[],LB,UB,nonlcon,IntCon,options)
x = ga(problem)
[x,fval] = ga(fitnessfcn,nvars,...)
[x,fval,exitflag] = ga(fitnessfcn,nvars,...)
[x,fval,exitflag,output] = ga(fitnessfcn,nvars,...)
[x,fval,exitflag,output,population] = ga(fitnessfcn,nvars,...)
[x,fval,exitflag,output,population,scores] = ga(fitnessfcn,nvars,...)
```

Description

`x = ga(fitnessfcn,nvars)` finds a local unconstrained minimum, `x`, to the objective function, `fitnessfcn`. `nvars` is the dimension (number of design variables) of `fitnessfcn`. The objective function, `fitnessfcn`, accepts a vector `x` of size 1-by-`nvars`, and returns a scalar evaluated at `x`.

`x = ga(fitnessfcn,nvars,A,b)` finds a local minimum `x` to `fitnessfcn`, subject to the linear inequalities `A` $\cdot$ `x` ≤ `b`. `fitnessfcn` accepts input `x` and returns a scalar function value evaluated at `x`.

`x = ga(fitnessfcn,nvars,A,b,Aeq,beq)` finds a local minimum `x` to `fitnessfcn`, subject to the linear equalities `Aeq` $\cdot$ `x` = `beq` as well as `A` $\cdot$ `x` ≤ `b`. (Set `A=[]` and `b=[]` if no inequalities exist.)

`x = ga(fitnessfcn,nvars,A,b,Aeq,beq,LB,UB)` defines a set of lower and upper bounds on the design variables, `x`, so that a solution is found in the range `LB` ≤ `x` ≤ `UB`. Use empty matrices for `LB` and `UB` if no bounds exist.

`x = ga(fitnessfcn,nvars,A,b,Aeq,beq,LB,UB,nonlcon)` subjects the minimization to the constraints defined in `nonlcon`. The function `nonlcon` accepts `x` and returns vectors `C` and `Ceq`, representing the nonlinear inequalities and equalities respectively. `ga` minimizes the `fitnessfcn` such that `C(x)` ≤ `0` and `Ceq(x) = 0`. (Set `LB=[]` and `UB=[]` if no bounds exist.)

`x = ga(fitnessfcn,nvars,A,b,Aeq,beq,LB,UB,nonlcon,options)` minimizes with the default optimization parameters replaced by values in the structure `options`, which can be created using the `gaoptimset` function.

`x = ga(fitnessfcn,nvars,A,b,[],[],LB,UB,nonlcon,IntCon)` requires that the variables listed in `IntCon` take integer values.

> **Note:** When there are integer constraints, `ga` does not accept linear or nonlinear equality constraints, only inequality constraints.

`x = ga(fitnessfcn,nvars,A,b,[],[],LB,UB,nonlcon,IntCon,options)` minimizes with integer constraints and with the default optimization parameters replaced by values in the `options` structure.

`x = ga(problem)` finds the minimum for `problem`, where `problem` is a structure.

`[x,fval] = ga(fitnessfcn,nvars,...)` returns `fval`, the value of the fitness function at `x`.

`[x,fval,exitflag] = ga(fitnessfcn,nvars,...)` returns `exitflag`, an integer identifying the reason the algorithm terminated.

`[x,fval,exitflag,output] = ga(fitnessfcn,nvars,...)` returns `output`, a structure that contains output from each generation and other information about the performance of the algorithm.

`[x,fval,exitflag,output,population] = ga(fitnessfcn,nvars,...)` returns the matrix, `population`, whose rows are the final population.

`[x,fval,exitflag,output,population,scores] = ga(fitnessfcn,nvars,...)` returns `scores` the scores of the final population.

Input Arguments

`fitnessfcn`

Handle to the fitness function. The fitness function should accept a row vector of length `nvars` and return a scalar value.

When the `'Vectorized'` option is `'on'`, `fitnessfcn` should accept a `pop`-by-`nvars` matrix, where `pop` is the current population size. In this case `fitnessfcn` should return a vector the same length as `pop` containing the fitness function values. `fitnessfcn` should not assume any particular size for `pop`, since `ga` can pass a single member of a population even in a vectorized calculation.

`nvars`	Positive integer representing the number of variables in the problem.
`A`	Matrix for linear inequality constraints of the form `A x` ≤ `b`. If the problem has `m` linear inequality constraints and `nvars` variables, then • `A` is a matrix of size `m`-by-`nvars`. • `b` is a vector of length `m`. **Note:** `ga` does not enforce linear constraints to be satisfied when the `PopulationType` option is `'bitString'` or `'custom'`.
`b`	Vector for linear inequality constraints of the form `A x` ≤ `b`. If the problem has `m` linear inequality constraints and `nvars` variables, then • `A` is a matrix of size `m`-by-`nvars`. • `b` is a vector of length `m`.
`Aeq`	Matrix for linear equality constraints of the form `Aeq x` = `beq`. If the problem has `m` linear equality constraints and `nvars` variables, then • `Aeq` is a matrix of size `m`-by-`nvars`. • `beq` is a vector of length `m`. **Note:** `ga` does not enforce linear constraints to be satisfied when the `PopulationType` option is `'bitString'` or `'custom'`.
`beq`	Vector for linear equality constraints of the form `Aeq x` = `beq`. If the problem has `m` linear equality constraints and `nvars` variables, then • `Aeq` is a matrix of size `m`-by-`nvars`. • `beq` is a vector of length `m`.
`LB`	Vector of lower bounds. `ga` enforces that iterations stay above `LB`. Set `LB(i)` = –`Inf` if `x(i)`is unbounded below.
`UB`	Vector of upper bounds. `ga` enforces that iterations stay below `UB`. Set `UB(i)` = `Inf` if `x(i)` is unbounded above.

`nonlcon`	Function handle that returns two outputs:
	`[c,ceq] = nonlcon(x)`
	`ga` attempts to achieve `c` ≤ `0` and `ceq` = `0`. `c` and `ceq` are row vectors when there are multiple constraints. Set unused outputs to `[]`.
	You can write `nonlcon` as a function handle to a file, such as
	`nonlcon = @constraintfile`
	where `constraintfile.m` is a file on your MATLAB path.
	To learn how to use vectorized constraints, see Vectorized Constraints.
	Note: `ga` does not enforce nonlinear constraints to be satisfied when the `PopulationType` option is set to `'bitString'` or `'custom'`.
	If `IntCon` is not empty, the second output of `nonlcon` (`ceq`) must be empty (`[]`).

For information on how ga uses nonlcon, see Nonlinear Constraint Solver Algorithm.

options — Structure containing optimization options. Create options using gaoptimset, or by exporting options from the Optimization Tool as described in Importing and Exporting Your Work in the Optimization Toolbox documentation.

IntCon — Vector of positive integers taking values from 1 to nvars. Each value in IntCon represents an x component that is integer-valued.

> **Note:** When IntCon is nonempty, Aeq and beq must be empty ([]), and nonlcon must return empty for ceq. For more information on integer programming, see Mixed Integer Optimization.

problem — Structure containing the following fields:

fitnessfcn	Fitness function
nvars	Number of design variables
Aineq	A matrix for linear inequality constraints
Bineq	b vector for linear inequality constraints
Aeq	Aeq matrix for linear equality constraints
Beq	beq vector for linear equality constraints
lb	Lower bound on x
ub	Upper bound on x
nonlcon	Nonlinear constraint function
rngstate	Optional field to reset the state of the random number generator
intcon	Index vector of integer variables
solver	'ga'
options	Options structure created using gaoptimset or the Optimization Tool

Create problem by exporting a problem from the Optimization Tool, as described in Importing and Exporting Your Work in the Optimization Toolbox documentation.

Output Arguments

x — Best point that ga located during its iterations.

fval — Fitness function evaluated at x.

`exitflag`

Integer giving the reason `ga` stopped iterating:

Exit Flag	Meaning
`1`	**Without nonlinear constraints** — Average cumulative change in value of the fitness function over `StallGenLimit` generations is less than `TolFun`, and the constraint violation is less than `TolCon`.
	With nonlinear constraints — Magnitude of the complementarity measure (see Definitions) is less than `sqrt(TolCon)`, the subproblem is solved using a tolerance less than `TolFun`, and the constraint violation is less than `TolCon`.
`2`	Fitness limit reached and the constraint violation is less than `TolCon`.
`3`	Value of the fitness function did not change in `StallGenLimit` generations and the constraint violation is less than `TolCon`.
`4`	Magnitude of step smaller than machine precision and the constraint violation is less than `TolCon`.
`5`	Minimum fitness limit `FitnessLimit` reached and the constraint violation is less than `TolCon`.
`0`	Maximum number of generations `Generations` exceeded.
`-1`	Optimization terminated by the output or plot function.
`-2`	No feasible point found.
`-4`	Stall time limit `StallTimeLimit` exceeded.
`-5`	Time limit `TimeLimit` exceeded.

When there are integer constraints, `ga` uses the penalty fitness value instead of the fitness value for stopping criteria.

`output`

Structure containing output from each generation and other information about algorithm performance. The `output` structure contains the following fields:

- `problemtype` — String describing the type of problem, one of:
  - `'unconstrained'`
  - `'boundconstraints'`
  - `'linearconstraints'`
  - `'nonlinearconstr'`
  - `'integerconstraints'`
- `rngstate` — State of the MATLAB random number generator, just before the algorithm started. You can use the values in `rngstate` to reproduce the output of `ga`. See Reproducing Your Results.
- `generations` — Number of generations computed.
- `funccount` — Number of evaluations of the fitness function.
- `message` — Reason the algorithm terminated.
- `maxconstraint` — Maximum constraint violation, if any.

`population`

Matrix whose rows contain the members of the final population.

`scores`

Column vector of the fitness values (scores for `integerconstraints` problems) of the final population.

Examples

Given the following inequality constraints and lower bounds

$$\begin{bmatrix} 1 & 1 \\ -1 & 2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 2 \\ 2 \\ 3 \end{bmatrix},$$
$$x_1 \geq 0, \quad x_2 \geq 0,$$

use this code to find the minimum of the `lincontest6` function, which is provided in your software:

```
A = [1 1; -1 2; 2 1];
b = [2; 2; 3];
lb = zeros(2,1);
[x,fval,exitflag] = ga(@lincontest6,...
    2,A,b,[],[],lb)

Optimization terminated: average change in
the fitness value less than options.TolFun.

x =
    0.6700    1.3310

fval =
   -8.2218

exitflag =
     1
```

Optimize a function where some variables must be integers:

```
fun = @(x) (x(1) - 0.2)^2 + ...
    (x(2) - 1.7)^2 + (x(3) - 5.1)^2;
x = ga(fun,3,[],[],[],[],[],[],[], ...
   [2 3]) % variables 2 and 3 are integers

Optimization terminated: average change
in the penalty fitness value less
than options.TolFun and constraint violation
is less than options.TolCon.

x =
    0.2000    2.0000    5.0000
```

Alternatives

For problems without integer constraints, consider using `patternsearch` instead of `ga`.

More About

expand all

Complementarity Measure

In the nonlinear constraint solver, the complementarity measure is the norm of the vector whose elements are $c_i\lambda_i$, where c_i is the nonlinear inequality constraint violation, and λ_i is the corresponding Lagrange multiplier.

Tips

- To write a function with additional parameters to the independent variables that can be called by `ga`, see Passing Extra Parameters in the Optimization Toolbox documentation.
- For problems that use the population type `Double Vector` (the default), `ga` does not accept functions whose inputs are of type `complex`. To solve problems involving complex data, write your functions so that they accept real vectors, by separating the real and imaginary parts.

Algorithms

For a description of the genetic algorithm, see How the Genetic Algorithm Works.

For a description of the mixed integer programming algorithm, see Integer ga Algorithm.

For a description of the nonlinear constraint algorithm, see Nonlinear Constraint Solver Algorithm.

- Genetic Algorithm
- Getting Started with Global Optimization Toolbox
- Optimization Problem Setup

See Also

gamultiobj | gaoptimset | patternsearch

Was this topic helpful? Yes No

- Acknowledgments
- Trademarks
- Patents
- Terms of Use