Previous topic
One Fifth Rule

Next topic
Controlling the Stopping Criteria: BI-POP CMA-ES

This Page
Show Source

Quick search
Go

Covariance Matrix Adaptation Evolution Strategy

The Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [Hansen2001] implemented in the `cma` module makes use of the generate-update paradigm where a population is generated from a strategy and the strategy is updated from the population. It is then straight forward to use it for continuous problem optimization.

As usual the first thing to do is to create the types and as usual we'll need a minimizing fitness and an individual that is a `list`. A toolbox is then created with the desired evaluation function.

```
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)

toolbox = base.Toolbox()
toolbox.register("evaluate", benchmarks.rastrigin)
```

Then, it does not get any harder. Once a `Strategy` is instantiated, its `generate()` and `update()` methods are registered in the toolbox for uses in the `eaGenerateUpdate()` algorithm. The `generate()` method is set to produce the created `Individual` class. The random number generator from numpy is seeded because the `cma` module draws all its number from it.

```
def main():
    numpy.random.seed(128)

    strategy = cma.Strategy(centroid=[5.0]*N, sigma=5.0, lambda_=20*N)
    toolbox.register("generate", strategy.generate, creator.Individual)
    toolbox.register("update", strategy.update)

    hof = tools.HallOfFame(1)
    stats = tools.Statistics(lambda ind: ind.fitness.values)
    stats.register("avg", numpy.mean)
    stats.register("std", numpy.std)
    stats.register("min", numpy.min)
    stats.register("max", numpy.max)

    algorithms.eaGenerateUpdate(toolbox, ngen=250, stats=stats, halloffame=hof)
```

[Hansen2001]	Hansen and Ostermeier, 2001. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation*