Relatório 8 - Evolução diferencial

1 Introdução

A optimização é o processo de buscar técnicas para seleção das melhores alternativas, tornar ótimo, alcançar os objetivos desejados. Essas técnicas buscadas são chamada de técnicas de otimização, e buscam minimizar ou maximizar uma função objetivo através da escolha dos valores de variáveis dentro de um conjunto possível visando encontrar uma solução ótima para um dado problema.

A optimização tem diversos métodos, como:

- O métodos determinísticos, nos quais é possível prever todos os passos do processo a partir de seu ponto de partida, eles sempre levam à mesma resposta a partir do mesmo ponto inicial.
- Os métodos estocásticos ou aleatórios, nos quais números aleatórios são sorteados no momento de execução do código. Ou seja, cada execução pode levar a um resultado diferente
- Os métodos do tipo gradiente, que são muito familiares para os engenheiros. Eles empregam a derivada da função para encontrar seu valor ótimo. Um grande problema dos métodos do tipo gradiente é que eles são otimizadores locais, ou seja, eles encontrar valores de mínimo ou máximo locais e não globais como desejado.

A evolução diferencial é um algoritmo que visa encontrar o mínimo ou máximo global com o uso de uma estratégia baseada na biologia. Nela cria-se uma geração g0 com vários indivíduos e na qual cada individuo é um vetor com as variáveis que se deseja optimizar, depois gera-se uma geração de mutação, na qual cada indivíduo é dado por um individuo do vetor original mais um fator F vezes a diferença de dois outros indivíduos aleatórios da geração original.

Após ter a geração de mutação formada o algoritmo gera uma terceira geração apelidada de geração trial. Essa geração é obtida com um crossover da geração g0 e da geração de mutação, esse crossover é feito ao se verificar para cada indivíduo se um número aleatório entre 0.0 e 1.0 é menor que um valor CR, caso seja o individuo da geração de mutação irá para a geração trial, e caso não seja o indivíduo que irá será o da geração g0.

Com a geração trial pronta o algorítimo compara os indivíduos da geração g0 e os da geração trial, o que produzir o melhor valor irá para a próxima geração. Esse processo se repete por quantas gerações forem necessárias.

Nesse relatório foi implementado o algoritmo de evolução diferencial para minimizar a função de Rosenbrock.

2 Objetivos

- Usar a evolução diferencial para encontrar o valor de entrada que produz o menor valor para a função de Rosenbrock usando os parâmetros providos pelo professor.

3 Desenvolvimento

Para realizar a implementação dos dois algoritmos foi utilizada a linguagem de programação Common Lisp com um paradigma primordialmente funcional. Primeiramente foi necessário implementar a função generate-inputs que gera uma serie de entradas com uma certa quantidade de variáveis e dentro de um valor mínimo e um máximo.

```
;; Generates a certain amount of inputs
;; Number, number, number, number -> List of lists
(defun generate-inputs (num-inputs num-vars min max)
  (loop
     for i from 1 upto num-inputs
     collect (loop
                for j from 1 upto num-vars
                collect (- (random max) (random (- min))))))
```

Depois foi necessário criar uma função para calcular o resultado da função de Rosenbrock, para isso foram necessárias as seguintes funções:

```
;; Calculates the Rosenbrock Function for a list of values
;; List -> number
(defun rosenbrock (list)
  (let ((result 0))
    (loop
       for i from 0 upto (1- (length list))
       if (not (eql i (1- (length list))))
       do (setf result (+ result
             (single-rosenbrock (nth i list) (nth (1+ i) list))))
       else
```

```
      return result)))
;; Calculates the Rosenbrock Function for a two values
;; Number, number -> number
(defun single-rosenbrock (x1 x2)
  (+ (expt (- 1 x1) 2) (* 100 (expt (- x2 (expt x1 2)) 2))))
```

Em seguida foi hora de criar uma função que retornasse o vetor das mutações, o resultado foi a função a seguir:

```
;; Produces a list of the mutated inputs
;; List of lists, number -> list of lists
(defun mutation (inputs F)
  (let ((size (length inputs)))
    (mapcar #'(lambda (input)
                (mapcar #'+ input
                        (mapcar #'(lambda (x) (* F x))
                                (mapcar #'-
                                        (nth (random size) inputs)
                                        (nth (random size) inputs)))))
            inputs)))
```

Em posse da função que gera o vetor das mutações foi hoje de criar a função que retornava o vetor trial com os crossovers:

```
;; Generates the trial vector, that is a crossover between the inputs
↪  and the mutation vector
;; List of lists, number, number -> list of lists
(defun trial-vector (inputs F CR)
  (mapcar #'(lambda (input mutation)
              (if (<= (random 1.0) CR) mutation input))
          inputs (mutation inputs F)))
```

Foi então possível criar a função de optimização de rosenbrock usando de todas as funções previamente criadas e de uma outra função chamada find-smallest-value, que dada uma geração retorna o individuo que produz o menor resultado para a função rosenbrock:

```
;; Optimizes the values and than finds the values that produce the
↪  smallest value of rosenbrock
;; List of lists
```

```
(defun optimize-rosenbrock (inputs F CR max-generations)
  (labels ((evolute (generation num-generation)
             (if (not (eql num-generation max-generations))
                 (evolute
                  (mapcar #'(lambda (gen final)
                              (if (< (rosenbrock gen) (rosenbrock
                              ↪  final)) gen final))
                          generation (trial-vector generation F CR))
                          ↪  (1+ num-generation))
                 (find-smallest-value generation))))
    (evolute inputs 0)))
;; Finds the values that produce the smallest value of rosenbrock
;; List of lists -> list
(defun find-smallest-value (list)
  (labels ((rec (lst smallest)
             (if (not (null lst))
                 (rec (rest lst)
                      (if (< (rosenbrock (first lst)) (rosenbrock
                      ↪  smallest)) (first lst) smallest))
                 smallest)))
    (rec list (first list))))
```

Que para 100 valores de duas variáveis variando de -1.0 a 2.0, NP = 100, CR = 0.9, F = 0.5 e 1000 gerações produziu o seguinte resultado:

(1.0 1.0)

Ao aplicá-lo na função de Rosenbrock obtêm-se como resultado 0 que é menor valor que a função pode assumir.

4 Conclusão

Em suma, foi possível observar o poder do algoritmo evolucional, que foi implementado de maneira simples e objetiva e produz resultados muito satisfatórios. O algoritmo para qualquer número de variáveis encontra como resultado para cada variável o valor 1.0, e quando essas variáveis são aplicadas à função obtêm-se como resultado 0, que é o valor mínimo que a função pode assumir.

Ademais, por meio de testes foi possível observar que quanto maior o número de variáveis usadas maior é o número de gerações necessárias para produzir o valor ideal como pode-se observar nos resultados abaixo.

5 Resultados

Para todos os resultados aqui encontrados foram utilizados 100 valores de n variáveis variado de -1.0 a 2, NP = 100, CR = 0.9 e F = 0.5

5.1 2 variáveis e 100 gerações

(1.0019 1.0036)

5.2 5 variáveis e 100 gerações

(0.8077 0.6675 0.4417 0.2136 0.1012)

5.3 5 variáveis e 1000 gerações

(1.0000 1.0000 1.0000 1.0000 1.000)

5.4 10 variáveis e 1000 gerações

(0.9999 0.9993 0.9982 0.9955 0.9949 0.9898 0.9835 0.9671 0.9339 0.8720)

5.5 10 variáveis e 10000 gerações

(1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000)