Optimizing the Video Game Multi-Jump: Player Strategy, AI, and Level Design (by A. M. Broussard, M. E. Malandro, and A. Serreyn) errata

11/28/2017

Algorithm 1 was unfortunately mis-formatted in the final version of the paper. Here it is, correctly formatted, using python-ish notation.

Algorithm for computing the jump points $(x_1, \ldots, x_n)$ defining the multi-jump ending at $(d, \text{hmax}(d))$.

```
#Input: (f_1, ..., f_n), (c_1, ..., c_n), d, ε
#Output: (x_1, ..., x_n)

def NumericalInverse(f, y, a, b, e):
    g(x) = f(x) − y
    while True:
        try:
            return root(g(x), a, b, e)
        except RuntimeError (no root): #if there is no root on [a, b]
            b = b × 10 #then expand the search range to the right

def OptimalJumpPoints((f_1, ..., f_n), (c_1, ..., c_n), d, ε):
    e = ε
    def DerivativeCheck((x_1, ..., x_n)):
        for i = 1, ..., n:
            if |f_i'(x_i) − f_1'(x_1)| ≥ ε:
                return False
        return True
    F(x) = x − d + Σ_{i=2}^{n} NumericalInverse(f_i', f_1'(x), c_i, d, e/n)
    while True:
        x_1 = root(F(x), c_1, d, e/n)
        for i = 2, ..., n:
            x_i = NumericalInverse(f_i', f_1'(x_1), c_i, d, e/n)
        if |d − Σ_{i=1}^{n} x_i| < ε and DerivativeCheck((x_1, ..., x_n)):
            return True
        else:
            e = e × 0.1

return OptimalJumpPoints((f_1, ..., f_n), (c_1, ..., c_n), d, ε)
```