1 Overview

This recursive algorithm computes the first N Gram points for arbitrary N and to any a-priori given precision by solving equations of type $\theta(t) - (n-1)\pi = 0$ numerically. As we shall see, the algorithm relies heavily on the fact that for $t > 7$, $\theta(t)$ is a continuous, monotonic increasing function. The maximum number of recursion frames that this algorithm requires is $Log_2(sp^{-1})$, where p is the specified precision, and s is the initial value of a *step* parameter.

2 Strategy

The underlying idea behind this algorithm is to keep "jumping" over the unique solution to $\theta_n(t) := \theta(t) - (n-1)\pi = 0$ (i.e. the desired n-th Gram point), gradually decreasing the "jump" length (*step*) and thus converging to the solution. In the program code, θ_n is named *as_ exp_ theta*, which stands for *asymptotic expansion of θ-function*. In practice, the above strategy is implemented using recursion over *numerical_ solution_ recursion* function, with a unique recursion frame associated with each successful "jump". Each frame is provided with the x value that it must process (*curr_ x*) and its current *step* value. Then, this frame undertakes a series of updates of its x (in *step*-increments) in the direction of decreasing θ_n-function value. Provided that θ-function is continuous and monotonic increasing, this direction is always towards the solution. These updates are achieved by evaluating θ_n at the following two candidate values: *curr_ x* + *step* and *curr_ x* − *step*. After a certain number of these updates, x value must be less than one *step* away from the solution. This occurs when θ_n evaluated at one of the two candidates with respect to the current x-value changes sign. When this condition is met, our frame updates x value one last time and evokes itself with x as the new starting x value, (*curr_ x*), and *step*/2 as the new *step*. In context of this new recursion frame, these procedures are repeated, producing a nested sequence of frames that perform "jumping" over the Gram point by length that decreases as Log_2 with respect to the recursion depth. The recursion terminates when a *numerical_ solution_ recursion* function frame with *step* value smaller than precision (*prec*) is just about to jump over the solution, which implies that current x value is less than one *step* away from it. This ensures that this x is within the specified error tolerance *prec* from the actual n-th Gram point. Note that *step* parameter decreases with depth of recursion, which guarantees that such a frame will be evoked and, consequently, the algorithm will always terminate.

3 Minor Details

Previous section provides intuition about the strategy employed by this algorithm, while some minor implementation difficulties and details are omitted. First, recall that it is crucial that θ-function possesses certain properties that are only valid for, roughly speaking, $t > 7$. Thus, we must ensure that x values never get updated to values less than 7. This is accomplished by resetting x to 7 whenever it is about to go below this mark. Additionally, it should be noted that the *precision* requirement cannot be arbitrarily small. Even though theoretically the algorithm works for any positive value of *prec*, any attempt to actually implement it with sufficiently small *prec* on an actual machine will run into a problem of *stack overflow*—a condition when the maximum allowed recursion depth is reached. Given the maximum depth d allowed for a specific machine, the best possible precision that can be achieved is $1/2^d s$, where s is the initial step value. Based on the distribution of Gram points, s can be initialized to 1 or less. However, a too small initial *step* will significantly increase time required for the algorithm to terminate.