## A. Definition of Partial Computation

Formally, we can write $f_{i\to\ell}$, the partial computation from $v_i$ to $v_\ell$, via the recurrence:

$$f_{i\to\ell}(v_{0:i}) = \varphi_\ell\big(u_{ij}(v_{0:i})\big)_{j\in A_\ell},$$
$$u_{ij}(v_{0:i}) = \begin{cases} v_j & \text{if } j \le i \\ f_{i\to j}(v_{0:i}) & \text{if } j > i \end{cases}$$

Here, $u_{ij}(v_{0:i})$ will equal $v_j$ for all $j$. For $j \le i$, the value is obtained directly from the inputs to $f_{i\to\ell}$. For $j > i$, the value of $u_{ij}$ is computed according to the partial computation from $i$ to $j$.

## B. Proof of Proposition 3

*Proof.* We write $f_{0\to j}$ as a function of $\epsilon$ using the two given Taylor expansions:

$$\begin{aligned} f_{0\to j}(x+\epsilon) &= \varphi_j\big(f_{0\to k}(x+\epsilon)\big), \\ &= \varphi_j\Big(v_k + \big(f_{0\to k}(x+\epsilon) - v_k\big)\Big), \\ &= v_j + Q\Big(f_{0\to k}(x+\epsilon) - v_k\Big), \\ &= v_j + Q\big(R(\epsilon)\big), \end{aligned}$$

where we used the Taylor expansion of $\varphi_j$ in Line 3 and the Taylor expansion of $f_{0\to k}$ in Line 4. The last expression is a power series in $\epsilon$, and, since $f_{0\to j}$ is analytic (the composition of two analytic functions is analytic), it is necessarily the Taylor series expansion of $f_{0\to j}$ about $x$. □

## C. Truncated Forward Algorithm

The truncated forward algorithm is the following variant of the forward algorithm for discrete HMMs, where $N$ is an upper bound placed on the population size:

1. Set $\alpha_0(0) = 1$ and $\alpha_0(n) = 0$ for $n = 1$ to $N$.
2. For $k = 1$ to $K$
   (a) Compute the transition matrix $P_k$, where $P_k(n, n') = \Pr(n_k = n' \mid n_{k-1} = n)$ for all $n, n' \in \{0, \dots, N\}$ (details below)
   (b) For $n' = 0$ to $N$, set
   $$\alpha_k(n') = p(y_k \mid n') \sum_{n=0}^{N} \alpha_{k-1}(n) P_k(n, n')$$
3. The likelihood is $\sum_{n=0}^{N} \alpha_K(n)$

Step 2(b) takes $O(N^2)$ time. Step 2(a) may take $O(N^3)$ or $O(N^2 \log N)$ time, depending on how it is implemented. Observe that $n_k$ is the sum of two random variables:

$$n_k = z_k + m_k, \qquad z_k := \sum_{i=0}^{n_{k-1}} z_{k,i}$$

and we must reason about their convolution to construct the transition probabilities. Specifically, the $n$th row of $P_k$ is the convolution of the first $N$ values of the distribution $p(z_k \mid n_{k-1} = n)$ and the first $N$ values of $p(m_k)$:

$$P_k(n, n') = \sum_{z=0}^{N} \Pr(z_k = z \mid n_{k-1} = n) \Pr(m_k = n' - z)$$

Let us assume we can compute the first $N$ values of each distribution in $O(N)$ time. Then the time to compute each row of $P_k$ is $O(N^2)$ if we use the direct convolution formula above, but $O(N \log N)$ if we use the fast Fourier transform (FFT) for convolution, making the overall procedure either $O(KN^3)$ or $O(KN^2 \log N)$. While the FFT is superior in terms of running time, it is inaccurate in many cases (see Section 5).

```
def A(s, k):
    if k < 0: return 1.0

    # This allows constant to be constructed in log space
    const = GDual.const( y[k] * np.log(rho[k]) - gammaln(y[k] + 1), as_log=True )

    return (s**y[k]) * const * \
        diff( lambda u: Gamma(u, k), s*(1 - rho[k]), y[k] )

def Gamma(u, k):
    F = lambda u:   offspring_pgf( u, theta_offspring[k-1] )
    G = lambda u: immigration_pgf( u, theta_immigration[k] )
    return A(F(u), k-1) * G(u)

log_likelihood = log( A(1.0, K-1) )
```

*Figure 5.* Code for AD based forward algorithm.