SYNTHESIS USING CUBIC SPLINES

```
cubic_sinusoid_synth( phi_K0, phi_K1, theta_K0, theta_K1, H )
    # theta_K0, K1 ARE STARTING AND ENDING PHASE (RADIANS)
    # omega_K0, K1 ARE STARTING AND ENDING ANGULAR VELOCITY (RADIANS)
    # H IS DISTANCE BETWEEN START AND END IN SAMPLES
    M = np.round(( (theta_K0 + omega_K0*H - theta_K1) + H/2 * (omega_K1 - omega_K0) ) / (2*np.pi) )
    alpha_beta = np.array([
            [ 3/(H*H)   , -1/H ],
            [ -2/(H*H*H) , 1/(H*H) ]
    ])
    alpha_beta @= np.vstack( ( theta_K1 - theta_K0 - H*omega_K0 + 2*np.pi*M,
                               omega_K1 - omega_K0 ) )
    # alpha_beta should have shape (2, len(phi_K0))
    # polynomial (cubic) coefficients
    pcoefs = np.vstack( ( np.flipud(alpha_beta), omega_K0, theta_K0 ) )
    n = np.arange(H)
    theta_n = polyval(pcoefs, n)   # SEE 2) FOR POLYEVAL
    tracks = cos(theta_n)   # IDEALLY ACCELERATED USING TABLE LOOKUP.
                            # OR exp(j*theta_n)
```

IN AN OBJECT ORIENTED IMPLEMENTATION THIS CAN ALL BE EVALUATED AT STARTUP.

AMPLITUDE SCALAR OF SINUSOIDS

```
amp_ramp_synth( a_K0, a_K1, H )
    h = np.arange(H)/H   # CAN BE PRECOMPUTED IN OO IMPLEMENTATION.
    ret = a_K0 + (aK1 - a_K0)[:, None] * h
    return ret
    # ALTERNATIVELY TO AVOID MULTIPLIES, PARTIAL SUMMATION CAN BE DONE,
    # BUT COMPENSATION SHOULD BE DONE TO MINIMIZE ROUNDING ERRORS
```

SYNTHESIS USING CUBIC SPLINES

```
def polyval ( powers, z ):
    # powers HAS SHAPE (P, K) WHERE P IS NUMBER OF COEFFICIENTS IN
    # POLYNOMIAL AND K IS NUMBER OF POLYNOMIALS
    # Z HAS SHAPE (N,), THE NUMBER OF EVALUATIONS TO COMPUTE
    # RESULT HAS SHAPE (K, N)

    # EVALUATION USING HORNERS METHOD
    #  ...((( P_0 z + P_1 ) z + P_2 ) z ...
```

$\alpha_0 = P_0$

$\alpha_1 = \alpha_0 z + P_1 = P_0 z + P_1$

$\alpha_k = \alpha_{k-1} z + P_k = (P_0 z + P_1) z + P_2 = P_0 z^2 + P_1 z + P_2$

$\vdots$

$y[n] = y[n-1] z + x[n]$

$x = [P_0 \;\; P_1 \;\cdots\; P_{k-1}]$

$b = [1]$

$a = [-z]$

```
    p_z = signal.efilter ([1], [-z], powers)[-1]
```

OR, USING OUTER PRODUCTS

$$P = \begin{bmatrix} 4 & 3 & 2 \\ 5 & 4 & 3 \end{bmatrix}$$

$x = [3, 10, 15]$

WE WANT

$$\begin{bmatrix} \begin{bmatrix} 4(3)^2 + 3(3) + 2 & , & 4(10)^2 + 3(10) + 2 & , & 4(15)^2 + 3(15) + 2 \end{bmatrix} \\ \begin{bmatrix} 5(3)^2 + 4(3) + 3 & , & 5(10)^2 + 4(10) + 3 & , & 5(15)^2 + 4(15) + 3 \end{bmatrix} \end{bmatrix}$$

$$\alpha_1 = \begin{bmatrix} 4 \\ 5 \end{bmatrix}$$

$$\alpha_1 = \begin{bmatrix} 4 \\ 5 \end{bmatrix} \begin{bmatrix} 3 & 10 & 15 \end{bmatrix} + \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{matrix} 4(3)+3 & 4(10)+3 & 4(15)+3 \\ 5(3)+4 & 5(10)+4 & 5(15)+4 \end{matrix}$$

$$\alpha_2 = \begin{bmatrix} 4(3)+3 & 4(10)+3 & 4(15)+3 \\ 5(3)+4 & 5(10)+4 & 5(15)+4 \end{bmatrix} \begin{bmatrix} 3 & 10 & 15 \end{bmatrix} + \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

... SEE 25)

```
#SYNTHESIS USING CUBIC SPLINES
# polyeval (pcoefs, z)

    p_x = pcoefs[0][:, None]

    for p in pcoefs[1:]:
        p_x = p_x * z + p[:, None]

    return p_x
```