```
REM     QPM in TWJPA with 3WM: Gain vs. Signal Frequency
REM                    (c) 2021, A. Zorin
REM
REM     Method: Standard Runge-Kutta
REM     Grid: Coincides with natural nodes of the ladder circuit
REM     Parameters and Constants
REM     bL - beta_L in rf-SQUIDs
REM     beta - abs. value of the nonlinear coefficient
REM     M% - the main poling period
REM     M2% = M%/2 - one half of the poling period
REM     p = fp/f0, where fp - pump frequrncy, f0 - cutoff frequency
REM     J = p*(fp/fJ)^2, where fJ - the SQUID plasma frequency
REM     fs - signal frequency
REM     fi = fp - fs - idler frequency
REM     f1 = fp + fs - second "idler" frequency f_+
REM     f2 = fp + fi - third "idler" frequency f_-
REM     fp2= 2fp - second harmonic of the pump
REM     d = (2fs-fp)/fp = (fp-2fi)/fp - dimensionless detuning
REM     h0 = kp - ks - ki  - phase mismatch 0
REM     h1 = kp - k1 + ks  - phase mismatch I
REM     h2 = kp - k2 + ki  - phase mismatch II
REM     h3 = 2kp - k2p  - phase mismatch III
REM     h4 = k2 + ks - k2p - phase mismatch IV
REM     h5 = 2 * kp - k2p - phase mismatch V
REM     N% - the length of JTWPA array

DECLARE FUNCTION Trapezoid# (e#) ' The tapered meander shape
DECLARE FUNCTION KWave (v) ' Wavenumber
DECLARE FUNCTION Signum (v) ' Signum function

CONST pi = 3.1415926#
CONST Nmax% = 2001 ' should be always larger than N%

DIM SHARED apR#(0 TO Nmax%) ' Re Ap
DIM SHARED apI#(0 TO Nmax%) ' Im Ap
DIM SHARED asR#(0 TO Nmax%) ' Re As
DIM SHARED asI#(0 TO Nmax%) ' Im As
DIM SHARED aiR#(0 TO Nmax%) ' Re Ai
DIM SHARED aiI#(0 TO Nmax%) ' Im Ai
DIM SHARED a1R#(0 TO Nmax%) ' Re A_+
DIM SHARED a1I#(0 TO Nmax%) ' Im A_+
DIM SHARED a2R#(0 TO Nmax%) ' Re A_-
DIM SHARED a2I#(0 TO Nmax%) ' Im A_-
DIM SHARED ap2R#(0 TO Nmax%) ' Re A2p
DIM SHARED ap2I#(0 TO Nmax%) ' Im A2p
DIM SHARED sB%(0 TO Nmax%) ' local sign of coefficient beta

DIM SHARED f0, fJ ' cutoff and plasma frequencies, respectively
DIM SHARED Slope, Smoothing ' parameters of trapezoid b and a, respectively
Slope = 0.2
Smoothing = .15

N% = 1000
M2% = 224
M% = 2 * M2%

bL = 0.5
beta = 0.24 REM   Approx. corrresponds to the maximum beta value for bL = 0.5
Ap0 = 3.0 REM   Initial amplitude of pump
fp = 15 ' pump frequency in GHz
f0 = 90 ' cutoff frequency in GHz
fJ = 35 ' plasma frequency in GHz
f2p = 2 * fp
p = fp / f0
J = p * (fp / fJ) ^ 2

Counter% = 0
sign% = 1
412

OPEN "c:/output000.dat" FOR APPEND AS #1 ' preparing output into file
'GOTO 414

Swfm = 0
Sdig = 0
sB%(0) = 1
FOR j% = 1 TO Nmax% ' Quantizer: waveform => sB%(j%)
    ArgS# = j% / M2%
    'Tr# = Trapezoid#(ArgS#)
    'GOTO 432
    Swfm = Swfm + Trapezoid#(ArgS#) ' the trapezoid (tapered meander) integral
    'or
    'Swfm = Swfm + SIN(pi * ArgS#) ' the sine integral
    'or
    'Swfm = Swfm + SIN(pi * ArgS#) / SQR((SIN(pi * ArgS#) ^ 2 + 0.002)) ' the meander integral
    SdigP = Sdig + 1
    SdigM = Sdig - 1
    IF ABS(Swfm - SdigP) < ABS(Swfm - SdigM) THEN
        sB%(j%) = 1
        Sdig = Sdig + 1
    ELSE
        sB%(j%) = -1
        Sdig = Sdig - 1
    END IF
    432
    'PRINT j%, ArgS#, Trapezoid#(ArgS#) ', sB%(j%)
    'PRINT #1, ArgS#, Tr# ',Sdig, Swfm
    'sB%(j%) = 1
NEXT j%

414
'GOTO 214

Switch% = 1 ' normally = 1, but when 0, then modes f1 and f2 are switched off

dStart = -.997
dEnd = .997
dStep = (dEnd - dStart) / 700
d = -0.1 ' detuning, yet fixed
FOR d = dStart TO dEnd + 0.00001 STEP dStep ' sweep over full range of detuning, -1<d<1
    'd = -0.15

    fs = fp * (1 + d) / 2
    fi = fp * (1 - d) / 2
    f1 = fp + fs
    f2 = fp + fi

    kp = KWave(fp) ' calculating wavenumbers
    k2p = KWave(f2p)
    ks = KWave(fs)
    ki = KWave(fi)
    k1 = KWave(f1)
    k2 = KWave(f2)

    h = kp - ks - ki ' calculating phase mismatches
    h0 = kp - ks - ki ' Delta k_0
    h1 = kp + ks - k1 ' - Delta k_1
    h2 = kp + ki - k2 ' - Delta k_2
    h3 = k1 + ki - k2p ' - Delta k_3
    h4 = k2 + ks - k2p ' - Delta k_4
    h5 = 2 * kp - k2p ' - Delta k_5

    apR#(0) = Ap0 ' initial conditions
    apI#(0) = 0#
    asR#(0) = 0 '0.002#
    aInput# = 0.025#
    asI#(0) = aInput#
    aiR#(0) = 0#
    aiI#(0) = 0#
    a1R#(0) = 0#
    a1I#(0) = 0#
    a2R#(0) = 0#
    a2I#(0) = 0#
    ap2R#(0) = 0#
    ap2I#(0) = 0#

    FOR j% = 0 TO N% ' Solving CMEs for given d
        b = 0.5 * beta * sB%(j% + 0)

        cs0 = COS(h0 * j%)
        sn0 = SIN(h0 * j%)
        cs1 = COS(h1 * j%)
        sn1 = SIN(h1 * j%)
        cs2 = COS(h2 * j%)
        sn2 = SIN(h2 * j%)
        cs3 = COS(h3 * j%)
        sn3 = SIN(h3 * j%)
        cs4 = COS(h4 * j%)
        sn4 = SIN(h4 * j%)
        cs5 = COS(h5 * j%)
        sn5 = SIN(h5 * j%)

        cs005 = COS(h0 * (j% + 0.5))
        sn005 = SIN(h0 * (j% + 0.5))
        cs105 = COS(h1 * (j% + 0.5))
        sn105 = SIN(h1 * (j% + 0.5))
        cs205 = COS(h2 * (j% + 0.5))
        sn205 = SIN(h2 * (j% + 0.5))
        cs305 = COS(h3 * (j% + 0.5))
        sn305 = SIN(h3 * (j% + 0.5))
        cs405 = COS(h4 * (j% + 0.5))
        sn405 = SIN(h4 * (j% + 0.5))
        cs505 = COS(h5 * (j% + 0.5))
        sn505 = SIN(h5 * (j% + 0.5))

        cs01 = COS(h0 * (j% + 1))
        sn01 = SIN(h0 * (j% + 1))
        cs11 = COS(h1 * (j% + 1))
        sn11 = SIN(h1 * (j% + 1))
        cs21 = COS(h2 * (j% + 1))
        sn21 = SIN(h2 * (j% + 1))
        cs31 = COS(h3 * (j% + 1))
        sn31 = SIN(h3 * (j% + 1))
        cs41 = COS(h4 * (j% + 1))
        sn41 = SIN(h4 * (j% + 1))
        cs51 = COS(h5 * (j% + 1))
        sn51 = SIN(h5 * (j% + 1))

        ''''''''''''''''''''''''''''''''''''''''''''''
        apRj# = apR#(j%)
        apIj# = apI#(j%)
        ap2Rj# = ap2R#(j%)
        ap2Ij# = ap2I#(j%)
        aiRj# = aiR#(j%)
        aiIj# = aiI#(j%)
        asRj# = asR#(j%)
        asIj# = asI#(j%)
        a2Rj# = a2R#(j%)
        a2Ij# = a2I#(j%)
        a1Rj# = a1R#(j%)
        a1Ij# = a1I#(j%)

        GOSUB ComputeRHS

        k1pR# = fpR#
        k1pI# = fpI#
        k1p2R# = fp2R#
        k1p2I# = fp2I#
        k1iR# = fiR#
        k1iI# = fiI#
        k1sR# = fsR#
        k1sI# = fsI#
        k12R# = f2R#
        k12I# = f2I#
        k11R# = f1R#
        k11I# = f1I#

        ''''''''''''''''''''''''''''''''''''''''''''''
        cs0 = cs005
        sn0 = sn005
        cs1 = cs105
        sn1 = sn105
        cs2 = cs205
        sn2 = sn205
        cs3 = cs305
        sn3 = sn305
        cs4 = cs405
        sn4 = sn405
        cs5 = cs505
        sn5 = sn505

        apRj# = apR#(j%) + 0.5 * k1pR#
        apIj# = apI#(j%) + 0.5 * k1pI#
        ap2Rj# = ap2R#(j%) + 0.5 * k1p2R#
        ap2Ij# = ap2I#(j%) + 0.5 * k1p2I#
        aiRj# = aiR#(j%) + 0.5 * k1iR#
        aiIj# = aiI#(j%) + 0.5 * k1iI#
        asRj# = asR#(j%) + 0.5 * k1sR#
        asIj# = asI#(j%) + 0.5 * k1sI#
        a2Rj# = a2R#(j%) + 0.5 * k12R#
        a2Ij# = a2I#(j%) + 0.5 * k12I#
        a1Rj# = a1R#(j%) + 0.5 * k11R#
        a1Ij# = a1I#(j%) + 0.5 * k11I#

        GOSUB ComputeRHS

        k2pR# = fpR#
        k2pI# = fpI#
        k2p2R# = fp2R#
        k2p2I# = fp2I#
        k2iR# = fiR#
        k2iI# = fiI#
        k2sR# = fsR#
        k2sI# = fsI#
        k22R# = f2R#
        k22I# = f2I#
        k21R# = f1R#
        k21I# = f1I#

        ''''''''''''''''''''''''''''''''''''''''''''''
        apRj# = apR#(j%) + 0.5 * k2pR#
        apIj# = apI#(j%) + 0.5 * k2pI#
        ap2Rj# = ap2R#(j%) + 0.5 * k2p2R#
        ap2Ij# = ap2I#(j%) + 0.5 * k2p2I#
        aiRj# = aiR#(j%) + 0.5 * k2iR#
        aiIj# = aiI#(j%) + 0.5 * k2iI#
        asRj# = asR#(j%) + 0.5 * k2sR#
        asIj# = asI#(j%) + 0.5 * k2sI#
        a2Rj# = a2R#(j%) + 0.5 * k22R#
        a2Ij# = a2I#(j%) + 0.5 * k22I#
        a1Rj# = a1R#(j%) + 0.5 * k21R#
        a1Ij# = a1I#(j%) + 0.5 * k21I#

        GOSUB ComputeRHS

        k3pR# = fpR#
        k3pI# = fpI#
        k3p2R# = fp2R#
        k3p2I# = fp2I#
        k3iR# = fiR#
        k3iI# = fiI#
        k3sR# = fsR#
        k3sI# = fsI#
        k32R# = f2R#
        k32I# = f2I#
        k31R# = f1R#
        k31I# = f1I#

        ''''''''''''''''''''''''''''''''''''''''''''''
        cs0 = cs01
        sn0 = sn01
        cs1 = cs11
        sn1 = sn11
        cs2 = cs21
        sn2 = sn21
        cs3 = cs31
        sn3 = sn31
        cs4 = cs41
        sn4 = sn41
        cs5 = cs51
        sn5 = sn51

        apRj# = apR#(j%) + k3pR#
        apIj# = apI#(j%) + k3pI#
        ap2Rj# = ap2R#(j%) + k3p2R#
        ap2Ij# = ap2I#(j%) + k3p2I#
        aiRj# = aiR#(j%) + k3iR#
        aiIj# = aiI#(j%) + k3iI#
        asRj# = asR#(j%) + k3sR#
        asIj# = asI#(j%) + k3sI#
        a2Rj# = a2R#(j%) + k32R#
        a2Ij# = a2I#(j%) + k32I#
        a1Rj# = a1R#(j%) + k31R#
        a1Ij# = a1I#(j%) + k31I#

        GOSUB ComputeRHS

        k4pR# = fpR#
        k4pI# = fpI#
        k4p2R# = fp2R#
        k4p2I# = fp2I#
        k4iR# = fiR#
        k4iI# = fiI#
        k4sR# = fsR#
        k4sI# = fsI#
        k42R# = f2R#
        k42I# = f2I#
        k41R# = f1R#
        k41I# = f1I#

        ''''''''''''''''''''''''''''''''''''''''''''''

        apR#(j% + 1) = apR#(j%) + (k1pR# + 2 * k2pR# + 2 * k3pR# + k4pR#) / 6
        apI#(j% + 1) = apI#(j%) + (k1pI# + 2 * k2pI# + 2 * k3pI# + k4pI#) / 6

        ap2R#(j% + 1) = ap2R#(j%) + (k1p2R# + 2 * k2p2R# + 2 * k3p2R# + k4p2R#) / 6
        ap2I#(j% + 1) = ap2I#(j%) + (k1p2I# + 2 * k2p2I# + 2 * k3p2I# + k4p2I#) / 6

        aiR#(j% + 1) = aiR#(j%) + (k1iR# + 2 * k2iR# + 2 * k3iR# + k4iR#) / 6
        aiI#(j% + 1) = aiI#(j%) + (k1iI# + 2 * k2iI# + 2 * k3iI# + k4iI#) / 6

        asR#(j% + 1) = asR#(j%) + (k1sR# + 2 * k2sR# + 2 * k3sR# + k4sR#) / 6
        asI#(j% + 1) = asI#(j%) + (k1sI# + 2 * k2sI# + 2 * k3sI# + k4sI#) / 6

        a2R#(j% + 1) = a2R#(j%) + (k12R# + 2 * k22R# + 2 * k32R# + k42R#) / 6
        a2I#(j% + 1) = a2I#(j%) + (k12I# + 2 * k22I# + 2 * k32I# + k42I#) / 6

        a1R#(j% + 1) = a1R#(j%) + (k11R# + 2 * k21R# + 2 * k31R# + k41R#) / 6
        a1I#(j% + 1) = a1I#(j%) + (k11I# + 2 * k21I# + 2 * k31I# + k41I#) / 6


        ''''''''''''''''''''''''''''''''''''''''''''''

        apM = SQR(apR#(j% + 1) ^ 2 + apI#(j% + 1) ^ 2)
        asM = SQR(asR#(j% + 1) ^ 2 + asI#(j% + 1) ^ 2)
        aiM = SQR(aiR#(j% + 1) ^ 2 + aiI#(j% + 1) ^ 2) + .0000001
        a1M = SQR(a1R#(j% + 1) ^ 2 + a1I#(j% + 1) ^ 2) + .0000001
        a2M = SQR(a2R#(j% + 1) ^ 2 + a2I#(j% + 1) ^ 2) + .0000001
        ap2M = SQR(ap2R#(j% + 1) ^ 2 + ap2I#(j% + 1) ^ 2) + 0.000001
        '  P1out = a1M * a1M
        '  P2out = a2M * a2M
        gs = 20 * 0.4343 * LOG(asM / aInput#) ' Signal gain in dB
        gi = 20 * 0.4343 * LOG(aiM / aInput#) ' cross-gain in dB
        g1 = 20 * 0.4343 * LOG(a1M / aInput#)
        g2 = 20 * 0.4343 * LOG(a2M / aInput#)
        gp = 20 * 0.4343 * LOG(apM / aInput#)
        gp2 = 20 * 0.4343 * LOG(ap2M / aInput#)
        ' PRINT j% + 1, asM, aiM, a1M 'AsR#(j% + 1), AsI#(j% + 1), AiR#(j% + 1), AiI#(j% + 1)
        ' PRINT #1, j%, apM 'apM ', ap2M 'sB%(j%)
    NEXT j%
    PRINT d, gs, gi, gp ', g1, g2, gp2
    PRINT #1, (d + 1) / 2, gs ', gi  ' print into file
NEXT d

214
CLOSE
END

ComputeRHS:
''''''''''''''''''''''''''''''''''''''''''''''

fpR# = -b * ks * ki * (asRj# * aiRj# - asIj# * aiIj#) * cs0 * (ks + ki) / kp
fpR# = fpR# - b * ks * ki * (asRj# * aiIj# + asIj# * aiRj#) * sn0 * (ks + ki) / kp
fpR# = fpR# + b * ks * k1 * (asRj# * a1Rj# + asIj# * a1Ij#) * cs1 * (k1 - ks) / kp
fpR# = fpR# + b * ks * k1 * (asRj# * a1Ij# - asIj# * a1Rj#) * sn1 * (k1 - ks) / kp
fpR# = fpR# + b * ki * k2 * (aiRj# * a2Rj# + aiIj# * a2Ij#) * cs2 * (k2 - ki) / kp
fpR# = fpR# + b * ki * k2 * (aiRj# * a2Ij# - aiIj# * a2Rj#) * sn2 * (k2 - ki) / kp
fpR# = fpR# - b * kp * k2p * (apRj# * ap2Rj# + apIj# * ap2Ij#) * cs5 * (k2p - kp) / kp
fpR# = fpR# + b * kp * k2p * (apRj# * ap2Ij# - apIj# * ap2Rj#) * sn5 * (k2p - kp) / kp
fpI# = b * ks * ki * (asRj# * aiRj# - asIj# * aiIj#) * sn0 * (ks + ki) / kp
fpI# = fpI# - b * ks * ki * (asRj# * aiIj# + asIj# * aiRj#) * cs0 * (ks + ki) / kp
fpI# = fpI# - b * ks * k1 * (asRj# * a1Rj# + asIj# * a1Ij#) * sn1 * (k1 - ks) / kp
fpI# = fpI# + b * ks * k1 * (asRj# * a1Ij# - asIj# * a1Rj#) * cs1 * (k1 - ks) / kp
fpI# = fpI# - b * ki * k2 * (aiRj# * a2Rj# + aiIj# * a2Ij#) * sn2 * (k2 - ki) / kp
fpI# = fpI# + b * ki * k2 * (aiRj# * a2Ij# - aiIj# * a2Rj#) * cs2 * (k2 - ki) / kp
fpI# = fpI# - b * kp * k2p * (apRj# * ap2Rj# + apIj# * ap2Ij#) * sn5 * (k2p - kp) / kp
fpI# = fpI# - b * kp * k2p * (apRj# * ap2Ij# - apIj# * ap2Rj#) * cs5 * (k2p - kp) / kp

fp2R# = b * kp ^ 3 / k2p * (apRj# ^ 2 - apIj# ^ 2) * cs5
fp2R# = fp2R# + b * kp ^ 3 / k2p * 2 * apRj# * apIj# * sn5
fp2R# = fp2R# - b * ks * k2 * (a2Rj# * asRj# - a2Ij# * asIj#) * cs4 * (k2 + ks) / k2p
```

```
fp2R# = fp2R# + b * ks * k2 * (a2Rj# * asIj# + a2Ij# * asRj#) * sn4 * (k2 + ks) / k2p
fp2R# = fp2R# - b * ki * k1 * (a1Rj# * aiRj# - a1Ij# * aiIj#) * cs3 * (k1 + ki) / k2p
fp2R# = fp2R# + b * ki * k1 * (a1Rj# * aiIj# + a1Ij# * aiRj#) * sn3 * (k1 + ki) / k2p
fp2I# = -b * kp ^ 3 / k2p * (apRj# ^ 2 - apIj# ^ 2) * sn5
fp2I# = fp2I# + b * kp ^ 3 / k2p * 2 * apRj# * apIj# * cs5 'OK
fp2I# = fp2I# - b * ks * k2 * (a2Rj# * asRj# - a2Ij# * asIj#) * sn4 * (k2 + ks) / k2p
fp2I# = fp2I# - b * ks * k2 * (a2Rj# * asIj# + a2Ij# * asRj#) * cs4 * (k2 + ks) / k2p
fp2I# = fp2I# - b * ki * k1 * (a1Rj# * aiRj# - a1Ij# * aiIj#) * sn3 * (k1 + ki) / k2p
fp2I# = fp2I# - b * ki * k1 * (a1Rj# * aiIj# + a1Ij# * aiRj#) * cs3 * (k1 + ki) / k2p

fiR# = b * ks * kp * (asRj# * apRj# + asIj# * apIj#) * cs0 * (kp - ks) / ki
fiR# = fiR# - b * ks * kp * (asRj# * apIj# - asIj# * apRj#) * sn0 * (kp - ks) / ki
fiR# = fiR# + b * k2 * kp * (a2Rj# * apRj# + a2Ij# * apIj#) * cs2 * (k2 - kp) / ki
fiR# = fiR# + b * k2 * kp * (-a2Rj# * apIj# + a2Ij# * apRj#) * sn2 * (k2 - kp) / ki
fiR# = fiR# + b * k1 * k2p * (a1Rj# * ap2Rj# + a1Ij# * ap2Ij#) * cs3 * (k2p - k1) / ki
fiR# = fiR# + b * k1 * k2p * (a1Rj# * ap2Ij# - a1Ij# * ap2Rj#) * sn3 * (k2p - k1) / ki
fiI# = b * ks * kp * (asRj# * apRj# + asIj# * apIj#) * sn0 * (kp - ks) / ki
fiI# = fiI# + b * ks * kp * (asRj# * apIj# - asIj# * apRj#) * cs0 * (kp - ks) / ki
fiI# = fiI# + b * k2 * kp * (-a2Rj# * apIj# + a2Ij# * apRj#) * cs2 * (k2 - kp) / ki
fiI# = fiI# - b * k2 * kp * (a2Rj# * apRj# + a2Ij# * apIj#) * sn2 * (k2 - kp) / ki
fiI# = fiI# + b * k1 * k2p * (a1Rj# * ap2Ij# - a1Ij# * ap2Rj#) * cs3 * (k2p - k1) / ki
fiI# = fiI# - b * k1 * k2p * (a1Rj# * ap2Rj# + a1Ij# * ap2Ij#) * sn3 * (k2p - k1) / ki

fsR# = b * ki * kp * (aiRj# * apRj# + aiIj# * apIj#) * cs0 * (kp - ki) / ks
fsR# = fsR# - b * ki * kp * (aiRj# * apIj# - aiIj# * apRj#) * sn0 * (kp - ki) / ks
fsR# = fsR# + b * k1 * kp * (a1Rj# * apRj# + a1Ij# * apIj#) * cs1 * (k1 - kp) / ks
fsR# = fsR# + b * k1 * kp * (-a1Rj# * apIj# + a1Ij# * apRj#) * sn1 * (k1 - kp) / ks
fsR# = fsR# + b * k2 * k2p * (a2Rj# * ap2Rj# + a2Ij# * ap2Ij#) * cs4 * (k2p - k2) / ks
fsR# = fsR# + b * k2 * k2p * (a2Rj# * ap2Ij# - a2Ij# * ap2Rj#) * sn4 * (k2p - k2) / ks
fsI# = b * ki * kp * (aiRj# * apRj# + aiIj# * apIj#) * sn0 * (kp - ki) / ks
fsI# = fsI# + b * ki * kp * (-aiIj# * apRj# + aiRj# * apIj#) * cs0 * (kp - ki) / ks
fsI# = fsI# - b * k1 * kp * (a1Rj# * apRj# + a1Ij# * apIj#) * sn1 * (k1 - kp) / ks
fsI# = fsI# + b * k1 * kp * (-a1Rj# * apIj# + a1Ij# * apRj#) * cs1 * (k1 - kp) / ks
fsI# = fsI# - b * k2 * k2p * (a2Rj# * ap2Rj# + a2Ij# * ap2Ij#) * sn4 * (k2p - k2) / ks
fsI# = fsI# + b * k2 * k2p * (a2Rj# * ap2Ij# - a2Ij# * ap2Rj#) * cs4 * (k2p - k2) / ks

f2R# = -b * ki * kp * (aiRj# * apRj# - aiIj# * apIj#) * cs2 * (ki + kp) / k2
f2R# = f2R# + b * ki * kp * (aiRj# * apIj# + aiIj# * apRj#) * sn2 * (ki + kp) / k2
f2R# = f2R# + b * ks * k2p * (asRj# * ap2Rj# + asIj# * ap2Ij#) * cs4 * (k2p - ks) / k2
f2R# = f2R# + b * ks * k2p * (asRj# * ap2Ij# - asIj# * ap2Rj#) * sn4 * (k2p - ks) / k2
f2I# = -b * ki * kp * (aiRj# * apRj# - aiIj# * apIj#) * sn2 * (ki + kp) / k2
f2I# = f2I# - b * ki * kp * (aiRj# * apIj# + aiIj# * apRj#) * cs2 * (ki + kp) / k2
f2I# = f2I# - b * ks * k2p * (asRj# * ap2Rj# + asIj# * ap2Ij#) * sn4 * (k2p - ks) / k2
f2I# = f2I# + b * ks * k2p * (asRj# * ap2Ij# - asIj# * ap2Rj#) * cs4 * (k2p - ks) / k2

f1R# = -b * ks * kp * (asRj# * apRj# - asIj# * apIj#) * cs1 * (ks + kp) / k1
f1R# = f1R# + b * ks * kp * (asRj# * apIj# + asIj# * apRj#) * sn1 * (ks + kp) / k1
f1R# = f1R# + b * ki * k2p * (aiRj# * ap2Rj# + aiIj# * ap2Ij#) * cs3 * (k2p - ki) / k1
f1R# = f1R# + b * ki * k2p * (aiRj# * ap2Ij# - aiIj# * ap2Rj#) * sn3 * (k2p - ki) / k1
f1I# = -b * ks * kp * (asRj# * apRj# - asIj# * apIj#) * sn1 * (ks + kp) / k1
f1I# = f1I# - b * ks * kp * (asRj# * apIj# + asIj# * apRj#) * cs1 * (ks + kp) / k1
f1I# = f1I# - b * ki * k2p * (aiRj# * ap2Rj# + aiIj# * ap2Ij#) * sn3 * (k2p - ki) / k1
f1I# = f1I# + b * ki * k2p * (aiRj# * ap2Ij# - aiIj# * ap2Rj#) * cs3 * (k2p - ki) / k1
RETURN

FUNCTION KWave (v)
ffJ = v / fJ
ffJ2 = ffJ ^ 2
ff0 = v / f0
x = 0.5 * ff0 / SQR(1 - ffJ2) ' Argument arcsin
y = x / SQR(1 - x ^ 2) ' Argument arctan
KWave = 2 * ATN(y)
END FUNCTION

FUNCTION Signum (v)
v1 = v
av = ABS(v)
IF av < 0.0001 THEN
    Signum = 0
    GOTO 812
END IF
Signum = v1 / av
812
END FUNCTION

FUNCTION Trapezoid# (e#)
eR# = e# - INT(e#)
IF INT(e#) MOD 2 = 0 THEN
    signSt% = 1
ELSE
```

```
    signSt% = -1
END IF
ArgEa# = -eR# / Slope / Smoothing
IF ArgEa# < -20 THEN
    Ea# = 0
    GOTO 354
END IF
Ea# = EXP(ArgEa#)
354
ArgEc# = -(1 - eR#) / Slope / Smoothing
IF ArgEc# < -20 THEN
    Ec# = 0
    GOTO 355
END IF
Ec# = EXP(ArgEc#)
355
Eb# = EXP(-1 / Smoothing)
Trapezoid# = -signSt% * Smoothing * LOG(Ea# + Eb# + Ec#)
END FUNCTION
```