

APPENDIX 1

Computer Listings

A1.1. Series Resonant Converter M(Q,F)

The following routine correctly computes the voltage conversion ratio M as a function of Q and F . It is assumed that the converter operates with linear resistive load R , and $Q=R_0 / R$. The algorithm and equations used are described in section 4.5.

```
function M(Q,F)
{computation of series resonant converter M, as a function of
  load Q and frequency F}
  define k,ksi,g,qgt,cgt,tgt,mr,k1
  g=PI()/F          {gamma}
  k=INT(1/F)
{check for mode}
  k1=INT(0.5+sqrt(0.25+Q*PI()/(2*F)))
  if k1>k
    {type k CCM}
    ksi=k+(1+(-1)^k)/2      {subharmonic number}
    qgt=Q*g/2              {intermediate variable}
    cgt=(cos(g/2))^2       {intermediate variable}
    tgt=(tan(g/2))^2       {intermediate variable}
    {mr contains the result M}
    if F=1/ksi
      mr=F                {tangent function may give unpredictable
        results at resonance and subharmonics}
    else
      mr=qgt/(ksi^4*tgt+qgt^2)*((-1)^(k+1)+SQRT(1+(ksi^2-
        cgt)*(ksi^4*tgt+qgt^2)/(qgt^2*cgt)))
    end if
  else
    {type k1 DCM}
    if k1/2=INT(k1/2)
      {even DCM}
      mr=2*k1*F/(PI()*Q)
    else
      {odd DCM}
      mr=1/k1
    end if
  end if
  return mr
end function
```

A1.2. Parallel Resonant Converter M(J,F)

The following routine computes the operating point of the parallel resonant converter. It works in both the continuous conduction mode and the discontinuous conduction mode, provided that $F \geq 0.5$. The CCM solution is a straightforward application of Eqs. (5-31) and (5-35). In the case of the discontinuous conduction mode, Eqs. (5-45) must be solved iteratively. Doing so is

not straightforward, and can become an exercise in numerical analysis methods. The routine below first finds the solution for $\xi = \alpha + \beta$. This is done by combining the third and fourth lines of Eq. (5-45) to obtain

$$G_f(\xi) = \cos \xi + 0.5 (2J - \gamma)^2 + 0.5 \xi^2 + (2J - \gamma) \xi + (2J - \gamma) \sin \xi + \xi \sin \xi - 1$$

This equation is solved iteratively. Once ξ is known, the other angles (α, β , and δ) can be evaluated directly, and then M can be found.

The routine first searches for the neighborhood of the root by simply evaluating G_f at regular intervals (note that ξ can take on values between 0 and γ), beginning at $\xi = \gamma$. Once G_f changes polarity, the routine uses Newton's method to converge quickly to the root.

```
function M(J,F)
{computation of parallel resonant converter M, as a function of load current J
and frequency F}
{equations are valid only for F>0.5}
    define phi, g, qgt, jcrit, mr, phicrit, ksi1, ksi2, ksi3, G1, G2, G3,
        convg, epsln, a, b, d, dc, ic, kc, Gc, dG1, jsc
    g=PI()/F    {gamma}
    {check for mode}
    if F<0.5
        return "invalid F"
    end if
    jcrit = Jcr(g)
    if j < jcrit
        {CCM}
        phi=Fphi(J,g)
        mr = (2/g)*(phi-sin(phi)/cos(g/2))
    else
        {DCVM}
        jsc=0.5*g
        {test whether J exceeds the short-circuit current; exit if it
        does}
        if J>jsc
            mr="error: I>short ckt current"
            return mr
            exit function
        end if
        {find neighborhood of root where convergence is likely}
        dc=0.1
        Gc=1
        while Gc>0
            kc=g
            Gc=Gf(kc,J,g)
            dc=dc*0.1
            ic=dc*g
            while ((Gc>0) and (kc>0))
                kc=kc-ic
                Gc=Gf(kc,J,g)
            end while
        end while
        ksi2=kc
        ksi1=kc+ic
        {iteration: Newton's method}
        convg=0.000001    {criterion to test convergence}
```

```

    epsln=0.1          {factor to reduce gain of iteration algorithm}
    G1=Gf(ksi1,J,g)
    dG1=dGf(ksi1,J,g)

    while abs(G1)>convg      {iteration loop: ksi = alpha + beta}
        ksi3=ksi1-epsln*G1/dG1
        G1=Gf(ksi3,J,g)
        dG1=dGf(ksi3,J,g)
        ksi1=ksi3
    end while
    a = acos(0.5*(1+cos(ksi3)))
    b = ksi3-a
    d = g-b
    mr = 1+(2/g)*(J-d)
end if
return mr
end function

```

Other functions called by the function M(J,F)

```

function Gf(k,J,g)
{function required for iteration in DCVM}
    define r,u
    u=2*J-g
    r=cos(k)+k*sin(k)+0.5*(u^2+k^2)+u*(k+sin(k))-1
    return r
end function

```

```

function dGf(k,J,g)
{derivative of Gf, used for Newton iteration method}
    define r
    r=k*cos(k)+k+(2*J-g)*(1+cos(k))
    return r
end function

```

```

function Jcr(g)
{evaluation of Jcrit, the critical value of J. CCM for J<Jcrit, DCVM for J>Jcrit}
    define r
    r=-0.5*sin(g) + sqrt((sin(g*0.5))^2+0.25*(sin(g))^2)
    return r
end function

```

```

function Fphi(J,g)
{evaluation of the angle phi, needed for CCM solution}
    define r
    r = acos(cos(g/2)+J*sin(g/2))
    if g<pi()
        r=-r
    end if
    return r
end function

```

A1.3. Other functions for evaluation of stresses and other quantities, for the parallel resonant converter

The functions below evaluate the boundary values M_{C0} and J_{L0} . Also listed are routines for determination of peak stresses M_{CP} and J_{LP} , as well as functions which indicate the operating

mode and whether the converter operates with zero current or zero voltage switching at the given operating point. These functions require that M , J , and γ all be specified; it is intended that $M(J,F)$ be used first to solve iteratively for the operating point, and then the results used in the routines below. All functions operate correctly for both CCM and DCM. They use the relevant equations of Chapter 5.

Boundary values M_{C0} and J_{L0}

```
function Mco(M,J,g)
{evaluation of tank capacitor initial voltage Mco, given the steady state
solution
  M,J, and gamma}
define d,b,r,jcrit,phi
jcrit=Jcr(g)
if J>jcrit
  {DCM soln}
  d=J-(g/2)*(M-1)
  b=g-d
  r=1-cos(b)
else
  {CCM soln}
  phi=Fphi(J,g)
  r=-J*sin(phi)/cos(g*0.5)
end if
return r
end function
```

```
function JLo(M,J,g)
{evaluation of tank inductor initial current JLo, given the steady state
solution
  M,J, and gamma}
define d,b,r,jcrit,phi
jcrit=Jcr(g)
if J>jcrit
  {DCM soln}
  d=J-(g/2)*(M-1)
  b=g-d
  r=J+sin(b)
else
  {CCM soln}
  phi=Fphi(J,g)
  r=-(J^2-1)*tan(g*0.5)
end if
return r
end function
```

Peak stresses J_{LP} and M_{CP}

```
function JLPk(M,J,g)
{evaluation of peak tank inductor current JLPk, given the
steady state solution M,J, and gamma}
define d,b,r,jcrit,phi,jl,jl0
jcrit=Jcr(g)
if J>jcrit
  {DCM soln}
  d=J-(g/2)*(M-1)
```

```

        b=g-d
        if (g-d)<pi()/2
            r=JLo(M,J,g)
        else
            r=J+1
        end if
    else
        {CCM soln}
        phi=Fphi(J,g)
        jl0=JLo(M,J,g)
        if ((Mco(M,J,g)<1) and (jl0>0))
            r=jl0
        else
            j1=-sin(phi)/cos(0.5*g)
            r=J+sqrt((j1-J)^2+1)
        end if
    end if
    return r
end function

function MCpk(M,J,g)
{evaluation of peak tank capacitor voltage MCpk, given the
steady state solution M,J, and gamma}
define d,b,r,jcrit,phi,mc0,jl0,j1
jcrit=Jcr(g)
jl0=JLo(M,J,g)
if J>jcrit
    {DCM soln}
    d=J-(g/2)*(M-1)
    b=g-d
    if jl0>J
        mc0=Mco(M,J,g)
        r=sqrt((mc0+1)^2+(J-jl0)^2)-1
    else
        r=2
    end if
else
    {CCM soln}
    phi=Fphi(J,g)
    if jl0>J
        mc0=Mco(M,J,g)
        r=sqrt((mc0+1)^2+(J-jl0)^2)-1
    else
        j1=-sin(phi)/cos(0.5*g)
        r=1+sqrt((j1-J)^2+1)
    end if
end if
return r
end function

```

Operating mode

```

function ZCVS(M,J,g)
{determination of whether the converter operates with zero current
or zero voltage switching at the designated operating point}
define r,jl0
jl0=JLo(M,J,g)
if jl0<0
    r="ZCS"

```

```
        else
            r="ZVS"
        end if
        return r
    end function

function PRCmode(J,g)
{determination of operating mode}
    define jcrit,jsc,r
    if g>2*pi()
        return "mode unknown: F<0.5"
        exit function
    end if
    if J<0
        return "invalid: J<0"
        exit function
    end if
    jcrit=Jcr(g)
    jsc=0.5*g
    if J<jcrit
        return "CCM"
    else
        if J>jsc
            return "no soln: J>jsc"
        else
            return "DCVM"
        end if
    end if
end function
```