```
function [out1, out2] =
 Exzitonenanregung(Input,Method,Potential,Energies)
% Konstanten bestimmen
c                   = constants ;
me                  = 0.46*c.me ;
mh                  = 0.41*c.me ;
mu                  = (me*mh)/(me+mh) ;
C                   = - (c.e^2)/(8*pi^2*c.eps0*c.eps) ;

% Dimension der k Matrix und St#tzstellen f#r das phi-Intergral muss
 in
% Input Struktur gespeichert sein.
% z.B.:
%    Input = struct('I_k',  [0 1 10 100],   'N_k',  [10 10 10], ...
%                   'I_phi',[-pi pi],       'N_phi',[50])
[I_k, N_k, I_phi, N_phi] = deal(Input.I_k, Input.N_k, Input.I_phi,
 Input.N_phi) ;

[k  ,   g_k  ]      = integrate(I_k,N_k,4) ;
[phi,   g_phi]      = integrate(I_phi,N_phi,6) ;
dim                 = length(k) ;

% Erzeugen des 3D Gitters (k',k,phi)
[K1,K,PHI]          = meshgrid(k,k,phi);
 kk1                = K1(:,:,1) ;
 kk                 =  K(:,:,1) ;
weight_k            = repmat(g_k',[dim, 1]) ;

% Integration über Phi für beliebige Funnktionen F(k,k',phi)
weight_phi          = permute(repmat(g_phi,[1,dim,dim]),[3,2,1]) ;
PHIntegrate         = @(k,k1,phi,fcn)  sum((fcn(k,k1,phi) .*
 weight_phi), 3) ;

% g_c               : Konvergenzfaktor g(k,k')
% veff              : Integrand in Veff(k,k'); Veff(k,k') = veff dphi
 in [0,2pi]
% veff_ii           : Integrand in Diagonalelementen
% veff_ij           : Integrand in Nicht-Diagonalelementen
% t_ii              : Funktion f#r kinetische Energie

g_c                 = @(k,k1)       4*k.^4 ./(k.^2 +k1.^2).^2 ;
switch Potential
    case{'Coulomb'}
        veff        = @(k,k1,phi)  (C*k1   ./sqrt(k.^2
 +k1.^2-2*k.*k1.*cos(phi))).*(k~=k1);
        I           = 12.0015;
    case{'Keldysh'}
        % Hier muss das zum Keldyshpotential gehoerende veff/I rein
        veff        = 1 ;
        I           = 1;
    otherwise
end
```

```
veff_ij             = veff;
veff_ii             = @(k,k1,phi)   g_c(k,k1) .*veff_ij(k,k1,phi) ;
t_ii                = @(k,k1)       1/2/mu *(c.hbar*k).^2 .*eq(k,k1);

% Erstellen der verschiedenen Anteile der Hamiltionmatrix
T_ii                = t_ii(kk,kk1);
V_ii                = C*I*(kk==kk1).*kk -
 diag(PHIntegrate(K,K1,PHI,veff_ii)*g_k);
V_ij                = PHIntegrate(K,K1,PHI,veff_ij) .* weight_k;

% Hamiltonmatrix zusammenfuegen
H = T_ii + V_ij + V_ii ;

switch Method
    case{'Spektrum'}
        disp('Spektrum')
        out1        = [];
        b           = ones(dim,1) ;
        for E = Energies
        H_const     = (c.E_G - E - 1i*c.Gamma)*(kk==kk1) ;
        A           = H + H_const ;
        x_k         = linsolve(A,b) ;
        x           = 1/(2*pi)*g_k'*(k.*x_k);

        out1        = [out1 x] ;
        out2        = Energies ;
        end

    case{'Eigenwerte'}
        disp('Eigenwerte')
        % Bestimmung der Eigenwerte (eig_val) samt Normierung der
 Wellenfunktionen
        % (states). Beides beginnend mit dem Grundzustand (sort).
        [states, EW]   = eig(H,'vector');
        [EW, idx]      = sort(EW);
        states         = states(:,idx);
        norm           = sqrt(2*pi*(states.^2)'*(k.*g_k));
        for i=1:dim; states(:,i) = states(:,i)*1/
norm(i)*sign(states(1,i)); end

        % Anzeigen der Grundzustandsenergie
        disp(EW(1))
        out1            = EW ;
        out2            = states ;
end

end
```

*Published with MATLAB® R2016a*