```
clear all;

M = 100;
H = @(x,k) x^2*k^2;

function [N,L,x,k,dx,dk] = setup_grid(M)
  N = 2*M+1;
  L = sqrt(2*pi*N);
  dx = L/N;
  dk = 2*pi/L;
  x = (-M:1:M)*dx; #x = linspace(-M,M,N)*dx
  k = (-M:1:M)*dk; #k = linspace(-M,M,N)*dk
endfunction


function rv = dimGetAndCheck(x,y)
  # Checks if x,y are vectors of the same size (col/row does not matter).
  #
  if ~isvector(x) error("x: vector expected.") endif
  if ~isvector(y) error("y: vector expected.") endif
  [x0,x1] = size(x);
  [y0,y1] = size(y);
  N = 1 + abs(x1-x0);
  if (N == 1 + abs(y1-y0))
    rv = N;
  else
    error("vectors of equal size required.");
  endif
endfunction


function rv = FunctionMatrix(f,x,y)
  # FunctionMatrix(f,x,y)= matrix f(x_i,y_j)
  #
  N = dimGetAndCheck(x,y);
  F = eye(N);
  for m=1:N
    for n=1:N
      F(m,n) = f(x(m),y(n));
    endfor
  endfor
  rv = F;
endfunction


function rv = UCDFT(M)
  # Unitary Centered Discrete Fourier Transform (UCDFT)
  # M = (N-1)/2 : integer
  # Computes the NxN matrix
  #   UCDFT(M)= 1/sqrt(N)*[exp(-2*i*pi/N)^((m-M-1)*(n-M-1))]
  #
  N = 2*M+1;
  U = eye(N);
  w = exp(-2*i*pi/N);
  for m=1:N
    for n=1:N
      U(m,n)=w^((m-M-1)*(n-M-1));
    endfor
  endfor
  rv = U/sqrt(N);
endfunction


function checkL2funvec(phi,M)
  # phi has to be a L^2 normalized complex valued row vector with
  # phi(1)=phi(N)=0.
  #
  eps0 = 1e-9;
  if ~isvector(phi) error("vector expected.") endif
  if ~isrow(phi) error("row vector expected.") endif
  if (abs(norm(phi)-1.0)>eps0) error("norm(vector,2) is not 1.") endif
  if (length(phi)~=2*M+1) error("size of phi must be 2*M+1.") endif
  if ~(phi(1)==0 && phi(end)==0) error("boundary condition not satisfied") endif
endfunction


function rv = makePhi(f,x)
  # Create a row vector that will pass checkL2funvec from a function f
  # Ex. phi=makePhi(@(t) t^2,[1,2,3,4,5]) -> checkL2funvec(phi,2)
  fx = arrayfun(f,x);
  fx(1)=0;
  fx(end)=0;
  rv = fx/norm(fx);
endfunction


function [mu,nu] = projMeasures(phi,M)
  # Create the marginal measures mu,nu from the function-vector phi.
  checkL2funvec(phi,M);
  dftphi = UCDFT(M) * phi';
  psi = dftphi' ;
  mu=phi .* conj(phi);  # sum(mu)=sum(nu)=1 ? check
```

```
  nu=psi .* conj(psi);
endfunction


function rv = tensorProductMeasure(mu,nu)
  # g = mu \otimes nu => g(i,j)=mu(i)*nu(j)
  # Assumes that all checks have been done before
  rv = mu' .* nu;
endfunction


function rv = makeConstraintMatrix(M)
  # Build a matrix A for LP
  # Use A = makeConstraintMatrix(M) -> sparse A
  # full(A) builds normal reprsentation.
   flat = @(x)x(:);
   mkc = @(n)sparse( flat(repmat(1:n, [n 1])), ...
             flat(reshape(1:n*n,n,n) ), ...
             ones(n*n,1) );
   mkr = @(n)sparse( flat(repmat(1:n, [n 1])), ...
             flat(reshape(1:n*n,n,n)' ), ...
             ones(n*n,1) );
   mkA = @(n)[mkr(n);mkc(n)];
   rv = mkA(2*M+1);
endfunction


function [gamma,mu,nu,x,k,EK,ES,HM,phi,FMIN,ERRNUM] = SolveWithGLPK(H,phi,M)
  [N,L,x,k,dx,dk] = setup_grid(M);
  [mu,nu] = projMeasures(phi,M);
  tpm = tensorProductMeasure(mu,nu);
  HM = FunctionMatrix(H,x,k);
  C = HM(:);
  A = makeConstraintMatrix(M);
  b = [mu,nu](:);
  [XOPT, FMIN, ERRNUM, EXTRA]=glpk(C,A,b);
  gamma = reshape(XOPT,N,N);
  EK=trace(HM'*gamma);
  ES=trace(HM'*tpm);
endfunction


function checkGLPKSolution(g,m,n,ek,err,fmin)
  # Checks the results from "glpk".
  eps0 = 1e-7;
  gm = norm(sum(g,2)-m(:));
  gn = norm(sum(g,1)'-n(:));
  gp = full(sum(g(:)~=0));
  de = abs(fmin-ek);
  N = length(m);
  if (err~=0) fprintf("Warning: ERRNUM = %d \n",err); endif
  if (de > eps0) fprintf("Warning: FMIN=%d <> EK \n",fmin); endif
  if (gm>eps0) fprintf("Warning: px(gamma)- mu = %d \n",gm); endif
  if (gn>eps0) fprintf("Warning: pk(gamma)- nu = %d \n",gn); endif
  if (gp>2*N-1) fprintf("Warning: #supp(gamma) = %d \n",gp); endif
endfunction


function [gamma,mu,nu,x,k,EK,ES,HM,phi] = Solve1(H,f,M)
  # Solves EK(phi)=min{g: Tr(H'*g), px(g)=mu(phi), pk(g)=nu(phi)}
  # using LP method/function glpk.
  [N,L,x,k,dx,dk] = setup_grid(M);
  phi = makePhi(f,x);
  [gamma,mu,nu,x,k,EK,ES,HM,phi,FMIN,ERRNUM] = SolveWithGLPK(H,phi,M);
  # checks
  checkGLPKSolution(gamma,mu,nu,EK,ERRNUM,FMIN);
endfunction


function [gamma,mu,nu,x,k,EK,ES,HM,phi,FMIN] = SolveWithLinprog(H,phi,M)
  # [XOPT,FMIN,ITN] = linprog(A,b,c,k,maxit,tol)
  # max iterations ~ 50k (10k is too small)
  maxit = 50000;
  tol = 1e-9;
  [N,L,x,k,dx,dk] = setup_grid(M);
  [mu,nu] = projMeasures(phi,M);
  tpm = tensorProductMeasure(mu,nu);
  HM = FunctionMatrix(H,x,k);
  C = HM(:);
  A = makeConstraintMatrix(M);
  b = [mu(:);nu(:)];
  [XOPT, FMIN, ITN]=perform_linprog(A,b,C,0,maxit,tol);
  gamma = reshape(XOPT,N,N);
  EK=trace(HM'*gamma);
  ES=trace(HM'*tpm);
endfunction

function [gamma,mu,nu,x,k,EK,ES,HM,phi] = Solve2(H,f,M)
  # Solves EK(phi)=min{g: Tr(H'*g), px(g)=mu(phi), pk(g)=nu(phi)}
  # using LP method/function perform_linprog.
  [N,L,x,k,dx,dk] = setup_grid(M);
  phi = makePhi(f,x);
```

```
  [gamma,mu,nu,x,k,EK,ES,HM,phi,FMIN] = SolveWithLinprog(H,phi,M);
  # checks
  checkGLPKSolution(gamma,mu,nu,EK,0,FMIN);
endfunction

####
function energy = ESchroed(HM,U,M,phi,x,k)
  X = phi'/norm(phi);
  Y = (U * X')';
  mu = X .* conj(X);
  nu = Y .* conj(Y);
  tpm = tensorProductMeasure(mu,nu);
  energy = trace(HM*tpm');
endfunction


function constraint = g(phi)
  [a,b] = size(phi);
  constraint = [sumsq(phi)-1; phi(a); phi(b)];
endfunction


function [phi0, obj, info, iter, nf, lambda,x] = findGroundState(H,M)
  [N,L,x,k,dx,dk] = setup_grid(M);
  HM = FunctionMatrix(H,x,k);
  U = UCDFT(M);
  E = @(phi) ESchroed(HM,U,M,phi,x,k);
  X0 = makePhi(@(t) exp(-t^2/2),x);
  [phi0, obj, info, iter, nf, lambda] = sqp (X0, E, @g, []);
endfunction


function [HM,phi,E] = schroedEq(M,V,m,h)
  # Solve the Schroedinger equation by the FGH method
  [N,L,x,k,dx,dk] = setup_grid(M);
  VV = arrayfun(V,x);
  fac = h^2/(8*pi*m*N);
  HM = eye(N);
  for m=1:N
    for n=1:N
      if (m==n)
        HM(m,n) = fac*(N^2-1)/6 + VV(m);
      else
        HM(m,n) = fac*(-1)^(m-n)*cos(pi*(m-n)/N)/sin(pi*(m-n)/N)^2;
      endif
    endfor
  endfor
  [vv,eed] = eig(HM);
  [E,perm] = sort(diag(eed));
  phi = vv(:,perm);
endfunction

# Examples:
# [gamma,mu,nu,x,k,EK,ES,HM,phi] = Solve1(@(x,k) x^2*k^2,@(t) exp(-t^2/2),M);
# [gamma,mu,nu,x,k,EK,ES,HM,phi] = Solve1(@(x,k) x^2+k^2,@(t) exp(-t^2/2),M);
# [gamma,mu,nu,x,k,EK,ES,HM,phi] = Solve2(@(x,k) x^2*k^2,@(t) exp(-t^2/2) ,M);


### Ground states
# [phi0, obj, info, iter, nf, lambda,x] = findGroundState(@(x,k) x^2+k^2,M);
# plot(x,phi0)

# [phi0, obj, info, iter, nf, lambda,x] = findGroundState(H,M);
# plot(x,phi0)

#  H=@(x,k) int8(x^2+k^2>4);
# [gamma1,mu,nu,x,k,EK1,ES,HM,phi] = Solve1(H,@(t) exp(-t^2/2) ,M);
# [gamma2,mu,nu,x,k,EK2,ES,HM,phi] = Solve2(H,@(t) exp(-t^2/2) ,M);


#[N,L,x,k,dx,dk] = setup_grid(M);
#V = @(t) t^2;
#[HM,phi,E] = schroedEq(M,V,1/2,2*pi);  # hbar^2=2*m => m=1/2, h=2*pi
#g0 = @(t) exp(-t^2/2);
#h0 = arrayfun(g0,x);
#h0 = h0/norm(h0);
#plot(x,-phi(:,1),x,h0)
```