Name: Yifan Tian ID: 78921267

1.

(a)

```
function normsq1(A)
  l = size(A)[1]
  (m1,n,m2) = size(A[1])
  mpsl = zeros(m1,m1,m2,m2)
  for u1 = 1:m1, d1 = 1:m1, s = 1:n, mu1= 1:m2, md1 = 1:m2
   mpsl[u1,d1,mu1,md1] += A[1][u1,s,mu1]*A[1][d1,s,md1]
  end
  for i = 2:l
    mpsr = zeros(m1,m1,m2,m2)                                    #key difference with tensor
contraction, not herit.
    for u1 = 1:m1, d1 = 1:m1, s = 1:n, mu1= 1:m2, md1 = 1:m2, mui = 1:m2, mdi = 1:m2
      mpsr[u1,mui,mdi,d1] += mpsl[u1,mu1,md1,d1]*A[i][mu1,s,mui]*A[i][md1,s,mdi]
    end
    mpsl = mpsr
  end
  return mpsl[1,1,1,1]
end
normsq1(A) = 1
```

(b)

```
function normsq2(A)
  l = size(A)[1]
  (m1,n,m2) = size(A[2])
  mps1 = reshape(A[1],m1*m2,n)*reshape(A[1],n,m1*m2)             # put the contraction side
together.
  mps1 = reshape(mps1,m1*m1*m2,m2)
  for i = 2:l
    mps1d = mps1*reshape(A[i],m1,n*m2)
    mps2 = reshape(mps1d,m2,n*m1)*reshape(A[i],n*m1,m2)
    mps1 = reshape(mps2,m1,m1)
  end
  return mps1[1,1,1,1]
end
normsq2(A) = 1
```

(c)

```
function normsq3(A)
  l = length(A)
  A1 = A[1]
  @tensor begin
    mpsl[u1,d1,mu1,md1] := A1[u1,s,mu1]*A1[d1,s,md1]
  end
  for i = 2:l
```

Printed by Wolfram Mathematica Student Edition

```
    A1 = A[i]
  @tensor mpsl[u1,d1,mu1,md1] = mpsl[u1,d1,mu,md]*A1[mu,s,mu1]*A1[md,s,md1]
  end
  return mpsl[1,1,1,1]
end
normsq3(A) = 1
```

2.

```
 type MPS
  A
  oc::Int64
end
mp  = MPS(A,1)
```

move OC:
One tricky part: when do from right to left. The QR(A[n,m]) in julia always decompose in Q*R based on the order of argument in the function.

```
function moveto!(psi::MPS,i::Int64)
  if i > psi.oc                                                    # move OC from left
to right
    for n = psi.oc:i-1
      (ml,s,mr) = size(psi.A[n])
      intmps = reshape(psi.A[n],ml*s,mr)
      intq = qr(intmps)[1]; intr = qr(intmps)[2]                   # do QR decomposi-
tion
       if ml*s >= mr
         psi.A[n] = reshape(intq,ml,s,mr)                          # from Q to construct
left tensor
       else
         psi.A[n] = reshape(intq,ml,s,ml*s)
       end
      (ml_r,s_r,mr_r) = size(psi.A[n+1])
      if ml*s >= ml_r
        psi.A[n+1] = reshape(intr*reshape(psi.A[n+1],ml_r,s_r*mr_r),ml_r,s_r,mr_r)      # mutiply
right site with R
      else
        psi.A[n+1] = reshape(intr*reshape(psi.A[n+1],ml_r,s_r*mr_r),ml*s,s_r,mr_r)
      end
    end
  elseif i < psi.oc                                                # move OC from
right to left
    for n = psi.oc:-1:(i+1)
      (ml,s,mr) = size(psi.A[n])
      intmps = reshape(psi.A[n],mr*s,ml)                           #tricky part
      intq = qr(intmps)[1]; intr = qr(intmps)[2]
      if mr*s >= ml
        psi.A[n] = reshape(transpose(intq),ml,s,mr)                #transpose to keep right order of indices
```

```
    else
      psi.A[n] = reshape(transpose(intq),mr*s,s,mr)
    end
    (ml_l,s_l,mr_l) = size(psi.A[n-1])
    if mr*s > mr_l
      psi.A[n-1] = reshape(reshape(psi.A[n-1],ml_l*s_l,mr_l)*intr,ml_l,s_l,mr_l)
    else
      psi.A[n-1] = reshape(reshape(psi.A[n-1],ml_l*s_l,mr_l)*intr,ml_l,s_l,mr*s)
    end
   end
  end
  psi.oc = i
  return psi
end
```

3.

Idea of energybond program: move oc to the gauge position. use the feature of OC (orthogonality) to measure the operator.

```
function energybond(psi::MPS,i::Int64)
  l = size(psi.A)[1]
  psi = moveto!(psi,i)
  mptwo1 = psi.A[i]; mptwo2 = psi.A[i+1]
 @tensor begin
 szbond = scalar(mptwo1[m1,usz1,eu]*mptwo2[eu,usz2,m2]*Htwosite[usz1,usz2,dsz1,dsz2]*mpt-
wo1[m1,dsz1,ed]*mptwo2[ed,dsz2,m2])
  end
  return szbond
end
```

when we have energybond program, we can use the summation to calculate the energy of the system.

```
function energy_oc(psi::MPS)
  #calculate heisenberg energy using energybond
  l = size(psi.A)[1]
  energy = 0
  for i = 1:l-1
    energy += energybond(psi,i)
  end
  mp_norm = normsq3(psi)
  return energy/mp_norm
end
```

for product state A. the energybond is -0.25
energybond(A) = -0.25
The energy of a n-site product state is 0.25(n-1).

4. The idea of imaginary time evolution:

```
function TEBD(psi::MPS,nt,m)
  l = size(psi.A)[1]
  for t = 1:nt                                   # nt = β/τ, number of sweep
    psi = sweep(psi,m)
    newpsi = tensor_reverse(psi)
    newpsi = sweep(newpsi,m)
    psi = tensor_reverse(newpsi)
  end
  return psi
end

function tensor_reverse(psi::MPS)                #reverse the tensor to use dosvdtoright()
twice.
  l = size(psi.A)[1]
  newpsiA = 0*psi.A
  for i = l:-1:1
    psiAi = psi.A[i]
    (a,b,c) = size(psiAi)
    tensor_i = zeros(c,b,a)
    @tensor tensor_i[c,b,a] = psiAi[a,b,c]
    newpsiA[l-i+1] = tensor_i
  end
  newoc = l-psi.oc+1
  return MPS(newpsiA,newoc)
end

function sweep(psi::MPS,m)                       //sweep from head to tail.
  l = size(psi.A)[1]
    for i = 1:l-1
      Ai = psi.A[i]
      Ai1 = psi.A[i+1]
      @tensor begin
        AA[a,f,g,e] := Ai[a,b,c]*Ai1[c,d,e]*taugate[b,d,f,g]
      end
      (psi.A[i],psi.A[i+1])=dosvdtoright(AA,m)
    end
    psi.oc = l
    return psi
end

function dosvdtoright(AA,m::Int64)
  (a,b,c,d) = size(AA)
  AA = reshape(AA,a*b,c*d)
  mpsl = svd(AA)[1]
  spectrum = svd(AA)[2]
```

```
  right = svd(AA)[3]
  D = zeros(m,m)
  Dm = length(spectrum)
  if Dm >m                                                    #do truncation
   [D[i,i] = spectrum[i] for i = 1:m]
   right = D*transpose(right[:,1:m])
   return (reshape(mpsl[:,1:m],a,b,m), reshape(right,m,c,d))
  else                                                        #grow to the truncation cut-off m.
   [D[i,i] = spectrum[i] for i = 1:Dm]
   right = D[1:Dm,1:Dm]*transpose(right[:,1:Dm])
   return (reshape(mpsl[:,1:Dm],a,b,Dm), reshape(right,Dm,c,d))
  end
end
```

Plot of the time-evolution of 20-sites, $\beta = 1$, $\tau = 0.01$. nstep = $\frac{\beta}{\tau} = 100$