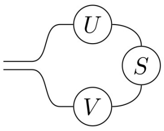
Exercise 8.1 (Schmidt decomposition and entanglement entropy)
Let $\psi \in \mathbb{C}^{m \cdot n}$ be a complex vector. We may interpret ψ as $m \times n$ matrix and compute its SVD, for which we use the convention $\psi = USV^T$ here, with isometries $U \in \mathbb{C}^{m \times k}$, $V \in \mathbb{C}^{n \times k}$ (such that $U^\dagger U = I$ and $V^\dagger V = I$), $k = \min(m, n)$ and $S = \operatorname{diag}(\sigma_1, \dots, \sigma_k)$. (The V matrix appears without complex conjugation in the SVD.) As graphical diagram:

ψ =



(a) Show that

$$\|\psi\|^2 = \sum_{j=1}^{k} \sigma_j^2.$$

Hint: Revisit the definition and properties of the Frobenius norm, or use the diagrammatic representation.

a) $\|\psi\|^2 \overset{①}{=} \|\psi\|_F^2 \overset{②}{=} \sqrt{\sum_j \sigma_j^2(\psi)}^{\,2} = \sum_j \sigma_j^2$ Vorlesung →

- Frobenius norm:

① $\|A\|_F = \sqrt{\sum_{i,j} |a_{ij}|^2} = \|\operatorname{vec}(A)\|$

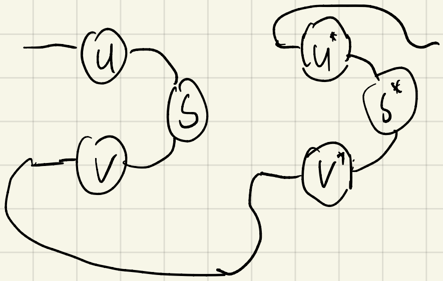
since $\sum_{i,j} |a_{ij}|^2 = \operatorname{tr}[A^\dagger A] = \sum_j \sigma_j^2(A)$

② $\|A\|_F = \sqrt{\sum_j \sigma_j^2(A)}$

(b) The partial trace has been introduced in Exercise 5.1. Here the two subsystems have dimension m and n, respectively. Verify that

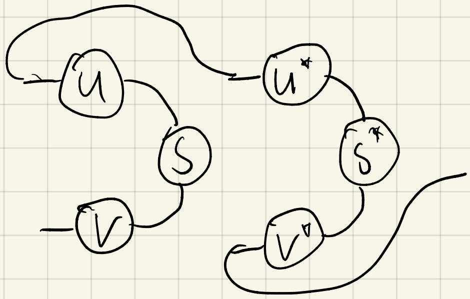
$$\operatorname{tr}_2[\psi \circ \psi^*] = US^2U^\dagger \quad \text{and}$$
$$\operatorname{tr}_1[\psi \circ \psi^*] = VS^2V^\dagger.$$

$\operatorname{tr}_2[\psi \circ \psi^*] =$

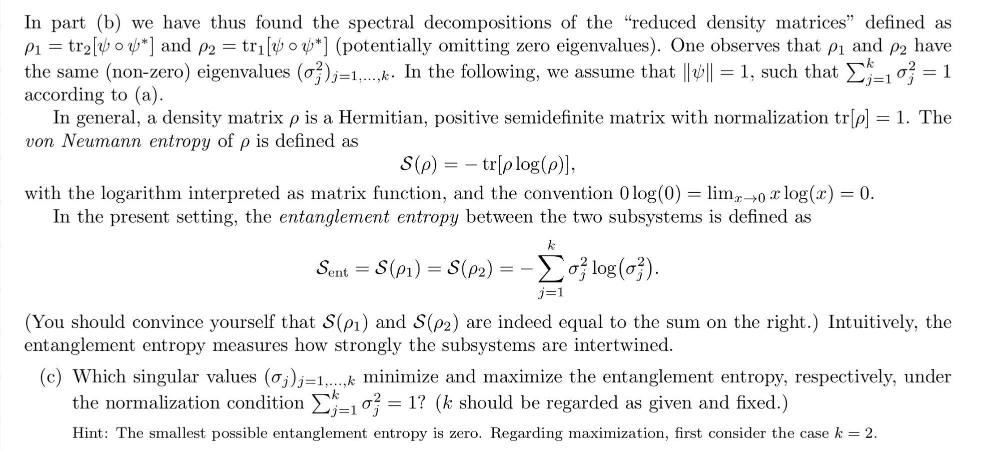


$= U \cdot S \cdot V^T \cdot V^* \cdot S^\dagger \cdot U^\dagger = US(VV^\dagger)^T S^* U^\dagger = USS^* U^T = US^2U^\dagger$

$\operatorname{tr}_1[\psi \circ \psi^*] =$



$= VS^T U^T U^* S^* V^\dagger = VSSV^\dagger = VS^2V^\dagger$



In part (b) we have thus found the spectral decompositions of the "reduced density matrices" defined as $\rho_1 = \mathrm{tr}_2[\psi \circ \psi^*]$ and $\rho_2 = \mathrm{tr}_1[\psi \circ \psi^*]$ (potentially omitting zero eigenvalues). One observes that ρ_1 and ρ_2 have the same (non-zero) eigenvalues $(\sigma_j^2)_{j=1,\dots,k}$. In the following, we assume that $\|\psi\| = 1$, such that $\sum_{j=1}^k \sigma_j^2 = 1$ according to (a).

In general, a density matrix ρ is a Hermitian, positive semidefinite matrix with normalization $\mathrm{tr}[\rho] = 1$. The *von Neumann entropy* of ρ is defined as

$$\mathcal{S}(\rho) = -\,\mathrm{tr}[\rho \log(\rho)],$$

with the logarithm interpreted as matrix function, and the convention $0 \log(0) = \lim_{x\to 0} x \log(x) = 0$.

In the present setting, the *entanglement entropy* between the two subsystems is defined as

$$\mathcal{S}_{\mathrm{ent}} = \mathcal{S}(\rho_1) = \mathcal{S}(\rho_2) = -\sum_{j=1}^{k} \sigma_j^2 \log(\sigma_j^2).$$

(You should convince yourself that $\mathcal{S}(\rho_1)$ and $\mathcal{S}(\rho_2)$ are indeed equal to the sum on the right.) Intuitively, the entanglement entropy measures how strongly the subsystems are intertwined.

(c) Which singular values $(\sigma_j)_{j=1,\dots,k}$ minimize and maximize the entanglement entropy, respectively, under the normalization condition $\sum_{j=1}^k \sigma_j^2 = 1$? ($k$ should be regarded as given and fixed.)

Hint: The smallest possible entropy entanglement entropy is zero. Regarding maximization, first consider the case $k = 2$.

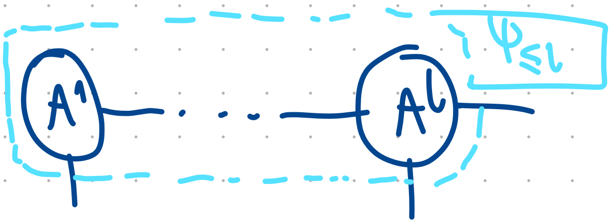
minimal entropy: $\sigma = (\underset{\substack{\uparrow \\ \text{beliebiger Index}}}{1}, 0, \dots, 0) \Rightarrow 1 \log(1) + (k-1) \cdot 0 \cdot \log(0) = \underline{\underline{0}}$

maximal entropy: distribution as even as possible:

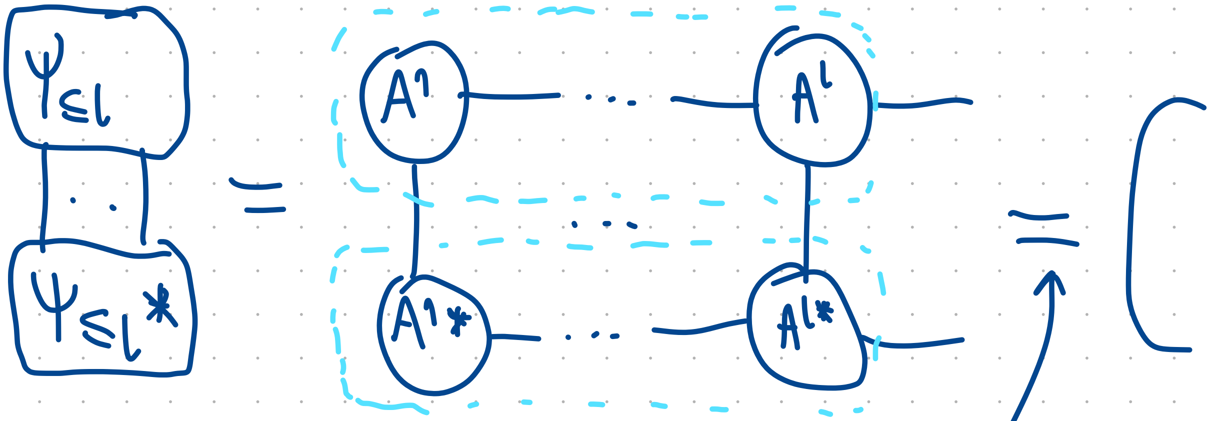
$$\sigma = \left(\tfrac{1}{\sqrt{k}}, \tfrac{1}{\sqrt{k}}, \dots\right) \Rightarrow S = -k \cdot \frac{1}{k} \cdot \log\left(\frac{1}{k}\right) = \underline{\underline{\log\left(\frac{1}{k}\right)}}$$

8.1d

- We have $\psi_{\leq l} =$

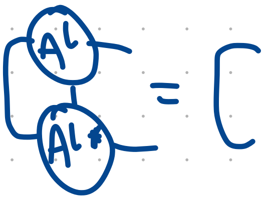


Then



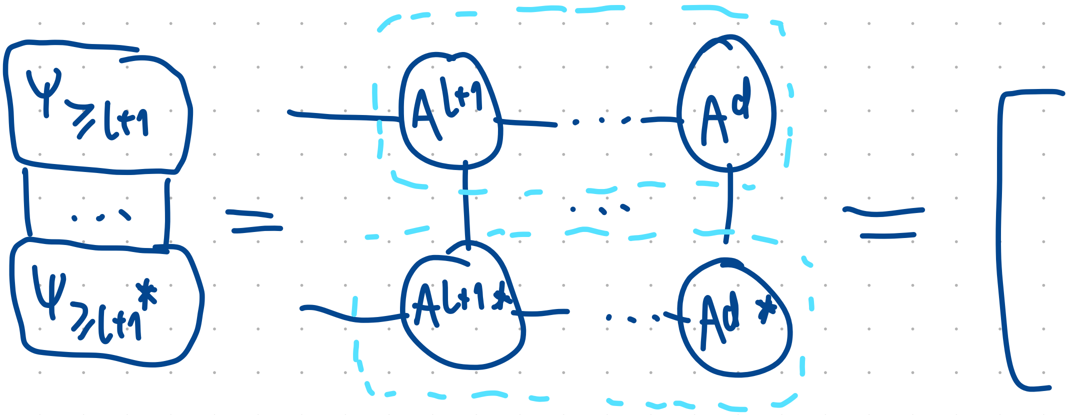
A^l are isometries

$\Rightarrow$ can apply



from lecture

- same for $\psi_{\geq l+1}$:



exercise8.2_template

June 22, 2022

1 Canonical MPS forms

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

1.1 Functions for conversion to canonical forms

[2]:
```
def mps_orthonormalize_left(Alist, stop=-1):
    """
    Left-orthonormalize a MPS using QR decompositions.
    The list of tensors in `Alist` are updated in-place.

    Returns the overall norm of the original MPS. (The updated MPS has norm 1.)
    """
    dummy = np.array([[[1]]])
    d = len(Alist)
    if (stop > 0):
        d = stop

    for l in range(d):
        n, dlm, dl = Alist[l].shape
        A = Alist[l].reshape((n * dlm, dl))
        Q, R = np.linalg.qr(A)
        dl = Q.shape[1]
        Alist[l] = Q.reshape((n, dlm, dl))
        if (l < len(Alist) - 1):
            Alist[l + 1] = np.moveaxis(np.tensordot(R, Alist[l + 1], ([1],␣
 ↪[1])), 0, 1)
        else:
            dummy = np.moveaxis(np.tensordot(R, dummy, ([1], [1])), 0, 1)

    dummy = dummy[0, 0, 0]
    if dummy < 0:
        dummy *= -1
        Alist[d - 1] = -1 * Alist[d - 1]

    return dummy
```

[3]:

```python
def mps_orthonormalize_right(Alist, stop=-1):
    """
    Right-orthonormalize a MPS using QR decompositions.
    The list of tensors in `Alist` are updated in-place.

    Returns the overall norm of the original MPS. (The updated MPS has norm 1.)
    """
    MirList = []

    for l in range(len(Alist) - 1, -1, -1):
        MirList.append(np.moveaxis(Alist[l], 1, 2))

    nrm = mps_orthonormalize_left(MirList, stop)

    for l in range(len(Alist)):
        Alist[len(Alist) - 1 - l] = np.moveaxis(MirList[l], 1, 2)

    return nrm
```

[4]:

```python
def mps_orthonormalize_center(Alist, j):
    """
    Convert a MPS to site-canonical form with center at site `j`, such that
    all tensors to the left are left-orthonormal, and
    all tensors to the right are right-orthonormal.
    The list of tensors in `Alist` are updated in-place.
    """
    mps_orthonormalize_right(Alist, stop=len(Alist) - j - 1)
    mps_orthonormalize_left(Alist, stop=j)
```

[5]:

```python
def mps_orthonormalize_bond(Alist, j):
    """
    Convert a MPS to bond-canonical form, with a list of "singular values"
    between the `j`-th and `j+1`-th tensor.
    The list of tensors in `Alist` are updated in-place.

    Returns the singular value list.
    """
    mps_orthonormalize_center(Alist, j)
    n, dlm, dl = Alist[j].shape
    A = Alist[j].reshape((n * dlm, dl))
    Q, R = np.linalg.qr(A)
    dl = Q.shape[1]
    Alist[j] = Q.reshape((n, dlm, dl))

    u, s, vh = np.linalg.svd(R)
    Alist[j] = np.tensordot(Alist[j], u, ([2], [0]))
    Alist[j+1] = np.moveaxis(np.tensordot(vh, Alist[j+1], ([1], [1])), 0, 1)
    return s
```

1.2 Utility functions

```
[6]: def is_left_orthonormal(A):
         """
         Test whether a MPS tensor `A` is left-orthonormal.
         """
         s = A.shape
         assert len(s) == 3
         A = np.reshape(A, (s[0] * s[1], s[2]))
         return np.allclose(A.conj().T @ A, np.identity(s[2]))
```

```
[7]: def is_right_orthonormal(A):
         """
         Test whether a MPS tensor `A` is right-orthonormal.
         """
         # call `is_left_orthonormal` with flipped left and right virtual bond␣
      ↪dimensions
         return is_left_orthonormal(np.transpose(A, (0, 2, 1)))
```

```
[8]: def mps_to_full_tensor(Alist):
         """
         Construct the full tensor corresponding to the MPS tensors `Alist`.

         The i-th MPS tensor Alist[i] is expected to have dimensions (n[i], D[i],␣
      ↪D[i+1]),
         with `n` the list of logical dimensions and `D` the list of virtual bond␣
      ↪dimensions.

         Note: Should only be used for debugging and testing.
         """
         # consistency check: dummy singleton dimension
         assert Alist[0].ndim == 3 and Alist[0].shape[1] == 1
         # formally remove dummy singleton dimension
         T = np.reshape(Alist[0], (Alist[0].shape[0], Alist[0].shape[2]))
         # contract virtual bonds
         for i in range(1, len(Alist)):
             T = np.tensordot(T, Alist[i], axes=(-1, 1))
         # consistency check: trailing dummy singleton dimension
         assert T.shape[-1] == 1
         # formally remove trailing singleton dimension
         T = np.reshape(T, T.shape[:-1])
         return T
```

[9]:

```
def mps_bond_to_full_tensor(Alist, S, j):
    """
    Construct the full tensor corresponding to the bond-canonical MPS
    with tensors `Alist` and "bond" singular values `S` between
    the `j`-th and `j+1`-th tensor.
    """
    # absorb bond singular values into j-th tensor
    Blist = [np.tensordot(Alist[i], np.diag(S), (2, 1)) if i == j else Alist[i]␣
  ↪for i in range(len(Alist))]
    return mps_to_full_tensor(Blist)
```

[10]:

```
def partial_trace(rho, dimA, dimB):
    """
    Compute the partial traces of a density matrix 'rho' of a composite quantum␣
  ↪system AB.

    Args:
        rho:  density matrix of dimension dimA*dimB x dimA*dimB
        dimA: dimension of subsystem A
        dimB: dimension of subsystem B
    Returns:
        tuple: reduced density matrices for subsystems A and B
    """
    # explicit subsystem dimensions
    rho = np.reshape(rho, (dimA, dimB, dimA, dimB))
    # trace out subsystem B
    rhoA = np.trace(rho, axis1=1, axis2=3)
    # trace out subsystem A
    rhoB = np.trace(rho, axis1=0, axis2=2)
    return rhoA, rhoB
```

[11]:

```
def crandn(size):
    """
    Draw random samples from the standard complex normal (Gaussian)␣
  ↪distribution.
    """
    # 1/sqrt(2) is a normalization factor
    return (np.random.normal(size=size) + 1j * np.random.normal(size=size)) /␣
  ↪np.sqrt(2)
```

[12]:

```
def xlogx(x):
    """
    Compute `x * log(x)` (pointwise), such that the result is zero for `x = 0`.
    """
    y = np.zeros_like(x)
    idx = x > 0
    y[idx] = x[idx] * np.log(x[idx])
    return y
```

1.3 Examples and tests

[13]:

```
# logical and virtual bond dimensions (rather arbitrarily chosen)
n = [2, 5, 3, 4, 6, 3]
D = [1, 3, 4, 7, 6, 5, 1]
```

[14]:

```
# random MPS tensors (the scaling factor keeps the norm of the full tensor in a␣
 ↪reasonable range)
np.random.seed(142)
Aref = [0.3 * crandn((n[i], D[i], D[i + 1])) for i in range(len(n))]

# the tensors are randomly chosen, and in particular not of any normal form
print([is_left_orthonormal(A) for A in Aref])
print([is_right_orthonormal(A) for A in Aref])

# construct the full (dense) tensor which this MPS represents, as reference␣
 ↪(should only be constructed for testing and debugging)
Tref = mps_to_full_tensor(Aref)
# its shape must be equal to `n` from above:
print("Tref.shape:", Tref.shape)
```

```
[False, False, False, False, False, False]
[False, False, False, False, False, False]
Tref.shape: (2, 5, 3, 4, 6, 3)
```

1.3.1 Left-orthonormalization

[15]:

```
# first make a copy of the input tensors
AL = [A.copy() for A in Aref]

# function returns norm of input MPS
nrmL = mps_orthonormalize_left(AL)
```

[16]:

```
# these should all be True
[is_left_orthonormal(A) for A in AL]
```

[16]:

```
[True, True, True, True, True, True]
```

[17]:

```
nrmL
```

[17]:

```
(1.2965535991355326-0j)
```

[18]:

```
# compare norm with reference
abs(nrmL - np.linalg.norm(np.reshape(Tref, -1))) / abs(nrmL)
```

[18]:
```
1.7125755932734126e-16
```

[19]:
```
# compare full tensor with reference: difference should be zero (up to␣
 ↪numerical rounding errors)
np.linalg.norm(nrmL * mps_to_full_tensor(AL) - Tref)
```

[19]:
```
1.1619396826252498e-15
```

1.3.2 Right-orthonormalization

[20]:
```
# first make a copy of the input tensors
AR = [A.copy() for A in Aref]

# function returns norm of input MPS
nrmR = mps_orthonormalize_right(AR)
```

[21]:
```
# these should all be True
[is_right_orthonormal(A) for A in AR]
```

[21]:
```
[True, True, True, True, True, True]
```

[22]:
```
nrmR
```

[22]:
```
(1.2965535991355326-0j)
```

[23]:
```
# compare norm with reference
abs(nrmR - np.linalg.norm(np.reshape(Tref, -1))) / abs(nrmR)
```

[23]:
```
1.7125755932734126e-16
```

[24]:
```
# compare full tensor with reference: difference should be zero (up to␣
 ↪numerical rounding errors)
np.linalg.norm(nrmR * mps_to_full_tensor(AR) - Tref)
```

[24]:
```
9.091941358304111e-16
```

1.3.3 Site-canonical form

[25]:
```
# again make a copy first
AC = [A.copy() for A in Aref]

# tensors are updated in-place, and overall norm is preserved (function has no␣
 ↪formal return value)
jcenter = 2
mps_orthonormalize_center(AC, jcenter)
```

[26]:

```
# these should all be True
[is_left_orthonormal(A) for A in AC[:jcenter]]
```

[26]:

```
[True, True]
```

[27]:

```
# these should all be True
[is_right_orthonormal(A) for A in AC[jcenter + 1:]]
```

[27]:

```
[True, True, True]
```

[28]:

```
# "center" tensor is not orthonormal in general
is_left_orthonormal(AC[jcenter]) or is_right_orthonormal(AC[jcenter])
```

[28]:

```
False
```

[29]:

```
# compare full tensor with reference: difference should be zero (up to␣
 ↪numerical rounding errors)
np.linalg.norm(mps_to_full_tensor(AC) - Tref)
```

[29]:

```
1.0487378834340819e-15
```

1.3.4 Bond-canonical form

[30]:

```
# again make a copy first
AB = [A.copy() for A in Aref]

jbond = 3
S = mps_orthonormalize_bond(AB, jbond)
```

[31]:

```
# list of singular values for "cut" at `jbond`
S
```

[31]:

```
array([0.85731111, 0.65318151, 0.47655434, 0.39473114, 0.27652958,
       0.24502551])
```

[32]:

```
# these should all be True
[is_left_orthonormal(AB[j]) if j <= jbond else is_right_orthonormal(AB[j]) for␣
 ↪j in range(len(AB))]
```

[32]:

```
[True, True, True, True, True, True]
```

[33]:

```
# compare full tensor with reference: difference should be zero (up to␣
 ↪numerical rounding errors)
np.linalg.norm(mps_bond_to_full_tensor(AB, S, jbond) - Tref)
```

[33]:

```
1.7067211880195904e-15
```

1.3.5 Bond-singular values and entanglement entropy

[34]:
```
# compute (reduced) density matrices, as reference
ref = np.outer(Tref, Tref.conj())
A, B = partial_trace( ref, np.prod(n[:jbond + 1]), np.prod(n[jbond + 1:]))
print(" A.shape:", A.shape)
print(" B.shape:", B.shape)
```

```
A.shape: (120, 120)
B.shape: (18, 18)
```

[35]:
```
# must be Hermitian
np.linalg.norm( A - A.conj().T)
```

[35]:
```
0.0
```

[36]:
```
# must be Hermitian
np.linalg.norm( B - B.conj().T)
```

[36]:
```
0.0
```

[37]:
```
A = np.linalg.eigvalsh( A)
B = np.linalg.eigvalsh( B)
```

[38]:
```
# most of them are actually zero
A
```

[38]:
```
array([-1.04158612e-16, -5.04120978e-17, -4.50643064e-17, -3.09628817e-17,
       -2.92487789e-17, -2.39908171e-17, -2.17945332e-17, -1.98333707e-17,
       -1.89664610e-17, -1.81748522e-17, -1.74202110e-17, -1.62433203e-17,
       -1.57288746e-17, -1.43154575e-17, -1.31744569e-17, -1.26977622e-17,
       -1.22028703e-17, -1.15573641e-17, -1.02748257e-17, -9.90329461e-18,
       -9.32509903e-18, -9.11640131e-18, -8.80149185e-18, -8.23987295e-18,
       -7.97671527e-18, -7.68367130e-18, -6.95259961e-18, -6.71073554e-18,
       -5.75320734e-18, -5.72178755e-18, -5.40778056e-18, -5.29059690e-18,
       -5.02807036e-18, -4.98842715e-18, -4.42073308e-18, -4.03270891e-18,
       -3.93415140e-18, -3.55608057e-18, -3.31606560e-18, -3.18667250e-18,
       -2.89392410e-18, -2.68396704e-18, -2.42062683e-18, -2.39460665e-18,
       -2.06096854e-18, -1.85741554e-18, -1.56622145e-18, -1.37198632e-18,
       -1.12395548e-18, -1.03684293e-18, -8.92166230e-19, -7.53258851e-19,
       -5.07123027e-19, -2.83237633e-19, -8.94118762e-20,  3.69924403e-21,
        4.17532460e-20,  1.20141516e-19,  2.61110534e-19,  3.85164442e-19,
        5.68331019e-19,  8.42177785e-19,  9.82611577e-19,  1.06496995e-18,
        1.20556263e-18,  1.33102835e-18,  1.53859718e-18,  1.63588470e-18,
        1.87965451e-18,  2.09521344e-18,  2.26005779e-18,  2.32445198e-18,
        2.51498965e-18,  2.90614325e-18,  3.07378425e-18,  3.24704011e-18,
        3.54241293e-18,  3.84657522e-18,  4.22021222e-18,  4.46631477e-18,
        4.63791846e-18,  5.12476071e-18,  5.27075754e-18,  5.67435143e-18,
       5.80687970e-18,  6.07796628e-18,  6.41923071e-18,  7.09885664e-18,
       7.31865144e-18,  8.01886322e-18,  8.41007922e-18,  8.81528485e-18,
       9.37137755e-18,  9.85957449e-18,  1.01493903e-17,  1.06423592e-17,
       1.17630356e-17,  1.19311664e-17,  1.27264057e-17,  1.33188034e-17,
       1.40625300e-17,  1.51276125e-17,  1.54795781e-17,  1.73981649e-17,
       1.83436600e-17,  2.02974773e-17,  2.20547510e-17,  2.32568787e-17,
       2.52686264e-17,  2.83287153e-17,  3.62885082e-17,  5.32190097e-17,
       6.48773579e-17,  9.00037809e-17,  6.00375012e-02,  7.64686066e-02,
       1.55812673e-01,  2.27104043e-01,  4.26646080e-01,  7.34982332e-01])
```

[39]:
```
# filter out zero eigenvalues
 A =  A[np.logical_not(np.isclose( A, 0, atol=1e-13))]
 B =  B[np.logical_not(np.isclose( B, 0, atol=1e-13))]

# sort in descending order
 A = np.sort( A)[::-1]
 B = np.sort( B)[::-1]
```

[40]:
```
 A
```

[40]:
```
array([0.73498233, 0.42664608, 0.22710404, 0.15581267, 0.07646861,
       0.0600375 ])
```

[41]:
```
# compare: should agree
np.linalg.norm( A -  B)
```

[41]:
```
3.632164660233972e-16
```

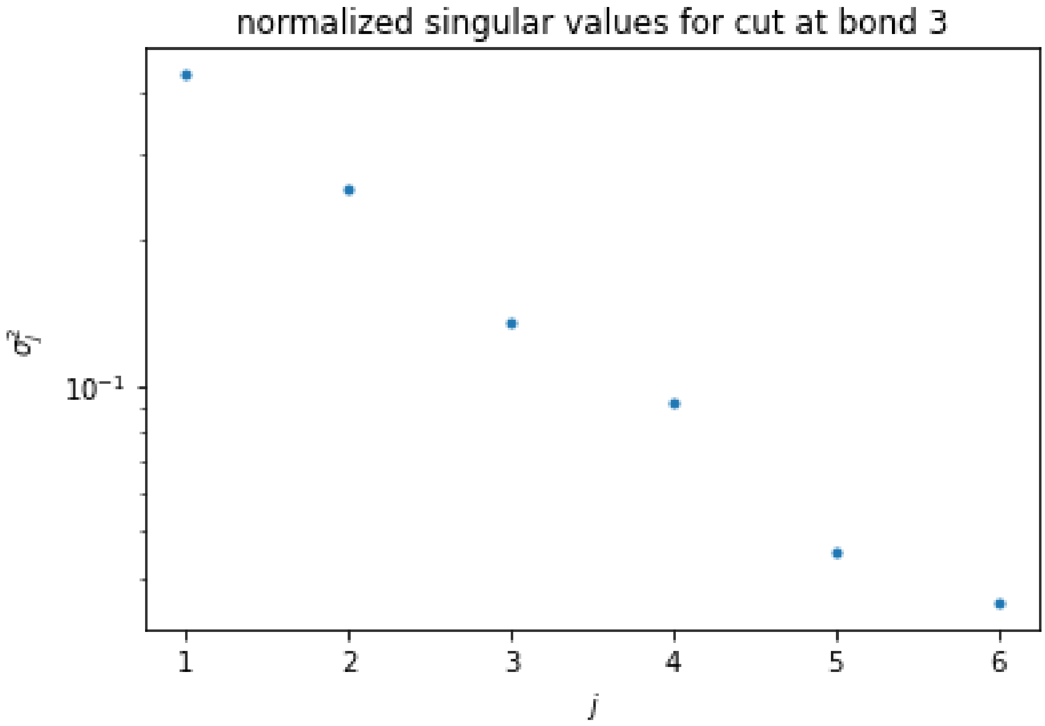
[42]:
```
# compare: should agree with bond-singular values from above
np.linalg.norm( A - S ** 2)
```

[42]:
```
7.132224383467466e-16
```

[43]:
```
# normalize singular values
Snrm = S / np.linalg.norm(S)
Snrm
```

[43]:
```
array([0.66122303, 0.50378288, 0.36755468, 0.30444645, 0.21328048,
       0.18898217])
```

[44]:
```
plt.semilogy(range(1, len(Snrm) + 1), Snrm ** 2, '.')
plt.ylabel("$\\sigma_j^2$")
plt.xlabel("$j$")
plt.title("normalized singular values for cut at bond {}".format(jbond))
plt.show()
```



[45]:
```
# finally compute entanglement entropy
np.sum(-xlogx(Snrm ** 2))
```

[45]:
```
1.460203986361991
```