Introduction to Quantics Tensor Train

Hiroshi SHINAOKA

Contents

1 はじめに

- Day 2 (1.5 hours × 3): Tensor networks/Quantics tensor trains

- Day 3 (1.5 hours × 3): Quantics tensor trains

この講義ノートでは, 以下の資料を参考にしている.
- [1]: 行列積表現
- [2]: quantics tensor train の簡単なレビューあり
- 「Quantics tensor train に基づく多スケール時空仮説と場の量子論」、品岡寛、村上雄太、野垣康介、櫻井理人、日本物理学会誌 2024 年 2 月号 「最近の研究から」[出版予定原稿](https://shinaoka.github.io/assets/qtt_jps_202402.pdf)

Google Colab を使用します. テンソルの数値計算には ITensors.jl を使用します. Julia を使用する際には, 「ランタイム」→「ランタイムのタイプを変更」→「ランタイムのタイプ」で Julia を選択してください. その後, ノートブックの最初のセルで以下のコードを実行してください.

```
using Pkg; Pkg.add("ITensors")
using ITensors
```

詳しい使い方は ITensors.jl 公式サイト https://itensor.org を参照すること.

2 線形代数の復習

2.1 行列の定義と基本的性質

行列は数や変数を長方形状に並べた二次元の配列で, 線形代数の中心的な対象である. 一般に m 行 n 列の行列は次のように表す.

$$A = (a_{ij}) \in \mathbb{R}^{m\times n} \tag{1}$$

または

$$A \in \mathbb{C}^{m\times n} \tag{2}$$

ここで a_{ij} は行 i, 列 j の成分を表し, 添字は $1 \le i \le m, 1 \le j \le n$ を満たす.

- 次元（shape）：行列 A のサイズは (m, n) と表す.
- 行ベクトル・列ベクトル：各行は長さ n の行ベクトル, 各列は長さ m の列ベクトルである.
- 転置：A^T は転置行列で, $(A^T)_{ij} = A_{ji}$ である.
- 単位行列：I_n は対角要素が全て 1 の $n \times n$ 行列で, 任意の $n \times n$ 行列 A に対して $I_n A = A I_n = A$ を満たす.
- 零行列：$0_{m\times n}$ は全成分が 0 の行列である.

2.2 行列の基本演算と計算量

- 加法：同じサイズの行列 $A, B \in \mathbb{R}^{m\times n}$ に対して $(A + B)_{ij} = A_{ij} + B_{ij}$.
  - 計算量：$O(mn)$（全要素に対する加算）
- スカラー倍：$\alpha \in \mathbb{R}$ に対して $(\alpha A)_{ij} = \alpha A_{ij}$.
  - 計算量：$O(mn)$（全要素に対する乗算）
- 行列積：$A \in \mathbb{R}^{m\times p}$, $B \in \mathbb{R}^{p\times n}$ のとき積 $C = AB \in \mathbb{R}^{m\times n}$ は

なお行列積は一般に可換ではない（多くの場合 $AB \neq BA$ である）.

- 計算量（単純な実装）：$O(mnp)$（各要素 C_{ij} を計算するのに $O(p)$ の和を取り, $m \times n$ 個の要素がある）
- 実用的には BLAS ライブラリによる最適化実装が利用される.

2.3 線形独立性（Linear Independence）

ベクトル空間 $\mathbb{R}^m$ において, ベクトルの集合 $\{\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_k\}$ が線形独立であるとは, 次の条件を満たすことである:

$$c_1\boldsymbol{v}_1 + c_2\boldsymbol{v}_2 + ... + c_k\boldsymbol{v}_k = \boldsymbol{0} \tag{4}$$

が成り立つのは　　の場合に限る.

言い換えると, どのベクトルも他のベクトルの線形結合では表せないということである. もし上記の等式が, 係数が全て 0 ではない（すなわち少なくとも一つの i について $c_i \neq 0$）場合に成り立つなら, その集合は線形従属（linearly dependent）である. 行列 $A \in \mathbb{R}^{m \times n}$ のランク（rank）は, 行または列の線形独立な最大個数として定義される. すなわち, 行空間の次元（行ランク）と列空間の次元（列ランク）は等しく, その共通の値を $\mathrm{rank}(A)$ と書く.

- 等価な定義：$\mathrm{rank}(A)$ は A の最大の線形独立な列の数であり, また A が写像として取りうる像（列空間）の次元に等しい.
- 性質：$\mathrm{rank}(A) \leq \min(m, n)$. 正方行列 $A \in \mathbb{R}^{n \times n}$ に対して $\mathrm{rank}(A) = n$ ならば A は可逆（逆行列を持つ）である.

ランクを計算する一般的な手法には特異値分解 (SVD), QR 分解, LU 分解 (ピボット付き) がある.

- 数値上の注意点：条件数（condition number）や丸め誤差により, 数値的にフルランクに見えても実際には近似的に低ランクである場合がある. 計算時には特異値の大きさの落ち方や閾値に基づいてランク判定を行うのが一般的である.

2.4 特異値分解（SVD）の基礎

実数行列 $A \in \mathbb{R}^{m \times n}$ に対して, SVD は次の形に分解される.

実数の場合: $A = U\Sigma V^{\mathsf{T}}$. 複素の場合: $A = U\Sigma V^{\dagger}$.

ここで $U \in \mathbb{R}^{m \times m}$（または $\mathbb{C}^{m \times m}$）, $V \in \mathbb{R}^{n \times n}$（または $\mathbb{C}^{n \times n}$）は直交（ユニタリ）行列, $\Sigma \in \mathbb{R}^{m \times n}$ は非負の対角成分のみを持つ対角形行列である.

2.4.1 直交（ユニタリ）性

- 実数: $U^{\mathsf{T}}U = I$, $V^{\mathsf{T}}V = I$.
- 複素: $U^{\dagger}U = I$, $V^{\dagger}V = I$.
- 解釈: U の列ベクトルは左特異ベクトルの正規直交基, V の列ベクトルは右特異ベクトルの正規直交基をなす.

2.4.2 特異値の基本性質

- 非負性: $\sigma_i \geq 0$.
- 降順整列: $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(m,n)} \geq 0$.
- ランクとの関係: $\mathrm{rank}(A) = |\{i \mid \sigma_i > 0\}|$. すなわち, 非零特異値の個数がランクに等しい.
- 条件数（2-ノルム）: $\kappa_2(A) = \frac{\sigma_1}{\sigma_{\min(m,n)}}$. これは行列の安定性の指標となる.
  - 注意: 下記の thin SVD では $r < \min(m,n)$ の場合, $\sigma_{\min(m,n)} = 0$ となり条件数は無限大となる.

2.4.3 一意性について

- 重複する特異値がない場合, 特異値は一意, 特異ベクトルは符号（実数）や位相（複素）の違いを除いて一意である.
- 重複がある場合, その等しい特異値に対応する部分空間内での直交変換の自由度が残る.

2.4.4 Thin SVD（経済的 SVD）

実際の計算では, ランク $r = \mathrm{rank}(A) \leq \min(m,n)$ を用いてより効率的な thin SVD が用いられる:

$$A = U_r \Sigma_r V_r^T \tag{5}$$

ここで:

- $U_r \in \mathbb{R}^{m \times r}$: 左特異ベクトル（正規直交）
- $\Sigma_r \in \mathbb{R}^{r \times r}$: 非零特異値の対角行列
- $V_r \in \mathbb{R}^{n \times r}$: 右特異ベクトル（正規直交）

2.5 低ランク近似と情報圧縮

2.5.1 低ランク近似の最適性

行列 A のランク k 近似 A_k は, フロベニウスノルム $\|A - A_k\|_F$ を最小化する. SVD を用いると:

$$A_k = U_k \Sigma_k V_k^T \tag{6}$$

ここで U_k, Σ_k, V_k は最初の k 個の特異値・特異ベクトルから構成される.

2.5.2 SVD を使った情報圧縮

- 元の行列: $m \times n$ 要素
- 近似行列: $(m + n + 1) \times k$ 要素
- 圧縮率: $\frac{k}{m+n}$ が小さいほど高圧縮

数値計算例 1: ヒルベルト行列

$n \times n$ ヒルベルト行列 $H_{ij} = \frac{1}{i+j-1}$ の特異値を計算し, 低ランク近似の精度を確認する.

数値計算例 2: 画像圧縮のシミュレーション

$m \times n$ の行列を画像データと見立て, SVD による低ランク近似で画像圧縮をシミュレートする.

3 テンソルネットワーク

3.1 テンソルの基礎知識

ここでテンソルとは単なる多次元配列である.

3.1.1 Row major order と Column major order

Row major order とは, 行を順番に並べる順番である. Column major order とは, 列を順番に並べる順番である.

例えば, 2 次元$N \times N$行列 A_{ij} において, row major order では, A_{ij} のインデックスの並びは

となり, column major order では, A_{ij} のインデックスの並びは

となる.

同様に, テンソルに対する row major order と column major order は, テンソルのインデックスの順番によって決まる. 3 次元テンソルA_{ijk}において, column major order では, インデックスの並びは次のとおりになる.

数値計算ライブラリや言語はデフォルトでどちらかを使う (切り替えられるものもある).

- Row major order: numpy
- Column major order: Fortran, Julia, Eigen3 (C++)

などである.

3.1.2 Unfolding

Unfolding とは, テンソルを行列として展開する操作である. 例えば, 4 次元テンソル A_{ijkl} において, $I = (i, j)$, $J = (k, l)$ とすると,

$$A_{ijkl} = A_{IJ} \tag{9}$$

と unfolding できる. ダイアグラム表記は次のようになる.

インデックスIの大きさ (取りうる値の数)は, iとjの大きさの積である. 同様に, インデックスJの大きさは, kとlの大きさの積である.

このとき, Iのインデックスの並びには, row/column major order のどちらかを使うことができる. 例えば, i, jの大きさをN_i, N_jとすると, row major order では, Iのインデックスの並びは

となり, column major order では, Iのインデックスの並びは

となる.

3.1.3 Permutation

Permutation とは, インデックスの並びを変える操作である. 例えば, 3 次元テンソルA_{ijk}の添字を並べ替えて, $B_{j,i,k} = A_{ijk}$ としたテンソル$B_{j,i,k}$を得る操作である. 基本的には, メモリ上での値の並べ替えである.

3.1.4 Contraction

Contraction とは, テンソルのインデックスを縮約する操作である. 例えば, 3 次元テンソルA_{ijk}と 2 次元テンソルB_{jl}の Contraction は, $C_{ikl} = \sum_j A_{ijk}B_{jl}$ となる. このとき, jのインデックスは縮約され, i, k, lのインデックスは残る.

また, 3 つ以上のテンソルの Contraction は, 複数のテンソルのインデックスを縮約する操作である. 例えば, 4 次元テンソルA_{ijkl}と 3 次元テンソルB_{jmn}と 2 次元テンソルC_{kp}の Contraction は,

となる. このとき, j, kのインデックスは縮約され, i, l, m, n, pのインデックスは残る. 通常, 複数のテンソルで共通して現れるインデックスに対して縮約をとるというルール (Einstein summation convention) を使う.

さて, contraction は数値計算ではどうやって行うのだろうか. 上記の summation を単純に実行する方法もあるが, 実際には行列積に直して, BLAS ライブラリを使って高速に計算することが多い.

例えば, $C_{ikl} = \sum_j A_{ijk}B_{jl}$の例では:
1. A_{ijk}を permute してA_{ikj}に並び替える
2. Unfolding して$A_{ik,j}$を得る
3. Unfolding して$B_{j,l}$を得る
4. 行列積$C_{ik,l} = A_{ik,j}B_{j,l}$を計算する
5. Unfolding を逆にしてC_{ikl}を得る

このように, contraction は基本的には行列積の組み合わせに分解できる. 各言語には, 最適化された contraction を行うためのライブラリが用意されている. 例えば, numpy では, `numpy.einsum` を使うことができる. Julia では, `ITensors.jl` が便利である.

最後に, 3つ以上のテンソルの contraction を取る場合, 2つずつ contraction を取ることを繰り返すことで計算できる. しかし, contraction をとる順番は, テンソルの数において指数的に増えるため, 最適化が必要である. 例えば, `numpy.einsum` では, 複数の contraction をとる順番を自動的に最適化することができる (Nが大きい場合でも, 近似的に良い contraction 順序を見つけることができる).

演習: テンソルの縮約

次のテンソルの縮約を permutation と行列積（2つずつの縮約）を使って計算する手順を見つけよう.

$$D_{ilmnp} = \sum_{j,k} A_{ijkl} B_{jmn} C_{kp} \tag{13}$$

ヒント: まずAとBの縮約を行い, その後Cとの縮約を行うとよい.

3.2 テンソルネットワークとダイアグラム表記

テンソルネットワークとは, 複数のテンソルの縮約をとったものである. しかし, テンソルの数が増えると, Einstein summation convention を使っても, 式が複雑になり, 理解が難しくなる. そこで, 物理分野では, ダイアグラム表記を使うことが多い. 図示すると Figure 1 のようになる.

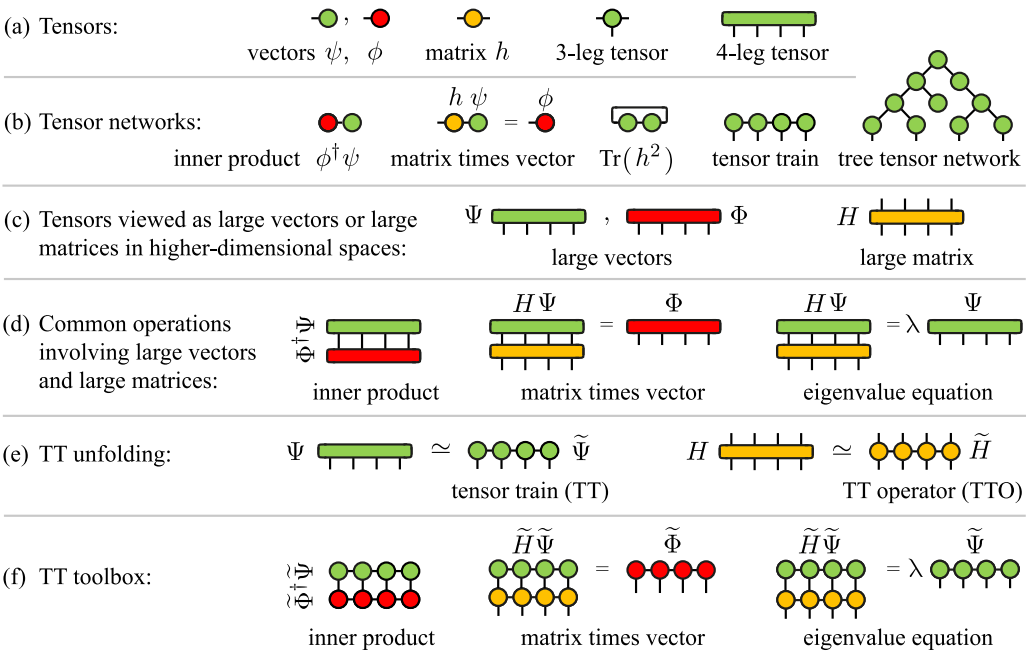


Figure 1: テンソルネットワークのダイアグラム表示と各種計算の表記法. SciPost Phys. 18, 104 (2025)より転載.

演習: ダイアグラム表記

Equation 12 をダイアグラム表記で表してみよう.

3.3 Tensor Train (TT)/Matrix Product State (MPS)

3.3.1 定義

Tensor train (TT)とは, Figure 1 の(e)のように, N次元テンソルをN個の 3 次元テンソルの積として表現した形式である. TT という呼び名は応用数学分野でよく使われるもので, 量子物理学の世界では, TT の発見以前から, Matrix Product State (MPS)と呼ばれている.

数式で書くと

$$F_{\boldsymbol{\sigma}} \approx \tilde{F} = \prod_{\ell=1}^{L} A_{\ell}^{\sigma_\ell} = [M_1]_{1a_1}^{\sigma_1} [M_2]_{a_1 a_2}^{\sigma_2} ... [M_L]_{a_{L-1} a_L}^{\sigma_L}. \tag{14}$$

$a_1, a_2, ..., a_L$は, 変数間の相関を表現する仮想的なインデックスである. それらの大きさをボンド次元と呼ぶことが多く, シンボルとしてはDやχが使われる. ダイアグラム表記では, 下記のようになる.

3.3.2 Evaluation

式 Equation 14 を見ると, 与えられたテンソルトレインを一つの$\boldsymbol{\sigma}$に対して評価する方法がわかる. 各テンソルコアM_ℓに対して, σ_ℓを代入すると, 3 次元テンソルから 2 次元行列になる. 両端のテンソルコアは, 片方のボンド次元が 1 であることから, ベクトルとみなせる. そのため, 端から行列・ベクトル積を計算することで, 最終的に 1 つの数値になる. ダイアグラムで表すと, 下記のようになる.

この計算量はボンド次元χに対して, $O(\chi^2 L)$である.

3.3.3 TT folding

与えられた多次元テンソルを分解して, テンソルトレインにする方法を考えよう. SVD を使う方法がもっともよく使われている. 他にも Tensor Cross Interpolation (TCI)という高速, ただし, ヒューリスティックな方法があるが, ここでは割愛する ([2] 参照).

SVD を使う場合, 多次元テンソルを片側から SVD を適用して, テンソルトレインにできる. ダイアグラムで表すと, 下記のようになる.

このとき, 途中の SVD で生じた特異値をまだ分解していない方にかけることが重要である. その結果, 得られたテンソルコアをA_ℓとすると, 左カノニカル形式になっている. 左カノニカル条件とは, 各テンソルコアA_ℓが次の正規直交性を満たすことである:

$$\sum_{\sigma_\ell} A_\ell^{\sigma_\ell \dagger} A_\ell^{\sigma_\ell} = I_{a_\ell} \tag{15}$$

カノニカル条件は, MPS の計算理論において, 極めて重要な役割を果たしている. しかし, 今回の講義の内容には深く関わるものではないため, ここでは割愛する. もし, 興味があれば, [1] を参照のこと.

ここでの SVD を用いた圧縮では, テンソルの全要素を読み出すため計算コストが高い. TCI では, 一部の要素のみを読み出すことで, 計算コストを低減する.

3.3.4 Recompression of TT

すでに与えられたテンソルトレインを再圧縮し, ボンド次元を小さくする事ができる. 詳しくは説明しないが, SVD の応用である. この際, 元のテンソルトレインFから, 圧縮後のテンソルトレイン$\tilde{F}$を得ることができる. 圧縮はそれらの間の距離を最小化するように行われる.

$$\tilde{F} = \mathrm{argmin}_{F^*} \|F - F^*\|_{\mathrm{F}}. \tag{16}$$

計算量は大まかに言って, $O(\chi^3 L^2)$程度である.

3.3.5 Tensor Train Operator (TTO)

次に, Tensor Train Operator (TTO)という概念を導入する. 物理では, Matrix Product Operator (MPO)とも呼ぶ.

簡単のため, 各次元が 2 つの値を取るテンソルトレインXを考えよう:

$$X_{\sigma_1 \sigma_2 \cdots \sigma_\ell \cdots \sigma_L} = [M_1]^{\sigma_1}_{1 a_1} [M_2]^{\sigma_2}_{a_1 a_2} \ldots [M_L]^{\sigma_L}_{a_{L-1} a_L}. \tag{17}$$

このテンソルトレインを 1 次元化し, サイズ2^Lの 1 次元ベクトルとみなす. そこに, 線形演算子W ($2^L \times 2^L$行列)を作用させることを考える:

$$Y_{\boldsymbol{\sigma}} = \sum_{\boldsymbol{\sigma}'} W_{\boldsymbol{\sigma}\boldsymbol{\sigma}'} X_{\boldsymbol{\sigma}'}. \tag{18}$$

この行列を TT 形式で表現すると, 下記のようになる.

ダイアグラムで書くと以下のようになる.

Equation 18 を見ると, テンソルトレインXを 1 次元化したベクトルに, 行列Wを作用させて, 新しいテンソルトレインYを得ることができる. これをダイアグラムとして表すと, 下記のようになる.

X, Wのボンド次元をχ, χ'とすると, 新しい TT のボンド次元は$\chi\chi'$となり, 計算コストは$O(\chi^2\chi'^2 L)$となる. 実際には, 計算結果は TT として再圧縮できることが多い. また, 一旦$\chi\chi'$のテンソルコアを得た後, それを SVD で圧縮することもできるが, 計算コストは高いため, 通常は行わない. より効率的なアルゴリズムとして, zip-up algorithm, fit algorithm (変分アルゴリズム)が存在するが, ここでは割愛する. これらのアルゴリズムを使い, 計算結果のボンド次元を$O(\chi)$まで圧縮する場合, 計算コストは大体$O(\chi^3\chi' L)$となることはここで覚えておくと良い (ただし, χ'はχと同程度かそれより小さい場合).

3.3.6 Tensor Train Operator (TTO)の応用

Tensor Train Operator (TTO)は, 物理では様々な場面で使われている. 例えば, 量子力学では, 量子状態をテンソルトレインで表現し, ハミルトニアンなどの線形演算子を TTO で表現することで, 量子力学の問題を解くことができる.

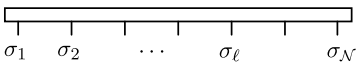
4 TCI と Quantics 表現の応用

4.1 関数のテンソル化

4.1.1 Natural representation

$\mathcal{N}$ 変数の関数 ($\boldsymbol{x} \in \mathbb{R}^{\mathcal{N}}$):

$$(\boldsymbol{x}(\boldsymbol{\sigma})) = f(x_1(\sigma_1), \cdots, x_{\mathcal{N}}(\sigma_{\mathcal{N}})) = \tag{20}$$



離散格子:

ここで, d_ℓ は ℓ 番目の変数の離散化格子のサイズである.

欠点: 局所構造と大きく異なる長さスケールが共存する場合, 局所構造を分解能するために d_ℓ を増やす必要がある.

4.1.2 Quantics representation

大きく異なる長さスケールが共存する場合に有効である [3,4].

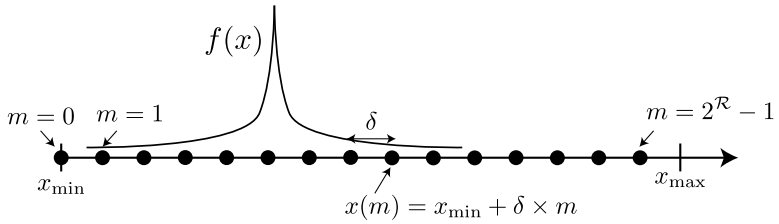
Binary coding

, (22)

ここで $a_r \in \{0,1\}$ である. 表現できる整数の範囲はビット数 $\mathcal{R}$ に対して指数的に増加する.

Quantics representation (1 変数)

関数 $f(x), x \in [x_{\min}, x_{\max}]$ を等間隔格子（サイズ $M = 2^{\mathcal{R}}$, 間隔 δ）上で離散化する:



サイズ $M = 2^{\mathcal{R}}$ の 1 脚テンソル:

$m = (\sigma_1 \cdots \sigma_{\mathcal{R}})_2$ を使うことで, 関数をサイズ $2 \times \cdots \times 2$ の $\mathcal{R}$ 脚テンソルとして表現できる:

Quantics representation ($\mathcal{N}$ 変数)

変数の 2 進表現を使う $(n = 1, \cdots, \mathcal{N})$: :

Interleaved representation:

Fused representation:

ボンド次元はインデックスの順序に強く依存する（下記参照）.

4.2 TT unfolding

Natural representation

$\mathcal{N}$ 変数関数を TT に展開（分解）する: $f(x_1, ..., x_{\mathcal{N}}) \approx M_1(x_1) M_2(x_2) ... M_{\mathcal{N}}(x_{\mathcal{N}})$,

TT が低ランクである場合, 超高速な和を実行できる:

$$\int d^{\mathcal{N}}\boldsymbol{x} f(\boldsymbol{x}) \approx \int dx_1 M_1(x_1) \int dx_2 M_2(x_2) ... \int dx_{\mathcal{N}} M_{\mathcal{N}}(x_{\mathcal{N}}). \tag{24}$$

実際には, 各変数に沿って Gauss 求積を使う. すなわち, 各変数を d_ℓ 個の Gauss-Legendre 節点に離散化する.

Quantics representation

TT unfolding は異なるスケール間の分離に対応する:

Riemann 和による積分:

$$\int dx f(x) \approx 2^{-\mathcal{R}} \left(\sum_{\sigma_1} M(\sigma_1) \right) \cdots \left(\sum_{\sigma_{\mathcal{R}}} M(\sigma_{\mathcal{R}}) \right) + O\left(2^{-\mathcal{R}}\right) \tag{25}$$

ブロックの塗りつぶされた円はベクトル $\left(\frac{1}{2}, \frac{1}{2}\right)$ である. $\mathcal{R}$ に関する線形計算コスト: $O(\chi^2 \mathcal{R})$, そして指数的に小さい離散化誤差.

Quantics 表現が適している場合:

- 多項式と指数関数: $e^{ax} =$.
- 単位行列: $\delta_{xy} =$.

Quantics 表現が適さない場合:

- $f(x, y) = 1$ if $x^2 + y^2 \leq 1$ otherwise 0. ボンド次元は $\mathcal{R}$ に対して指数的に増加する.

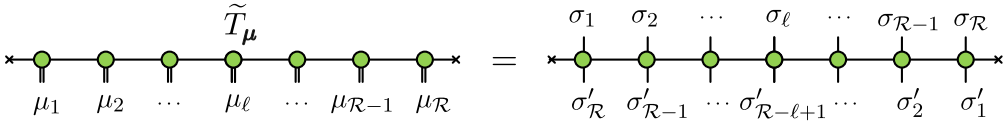
一般に, 超平面上の不連続性や急激な特徴は問題ないが, 超曲線上では問題となる.

4.3 QTT 形式での演算の例

4.3.1 Tensor Train Operator (TTO)との畳み込み

4.3.2 Quantum Fourier Transform (QFT)

MPO は小さなエンタングルメントエントロピーを持つ [5,6]:



σ_ℓ と σ'_ℓ はそれぞれ QFT の前後における変数のビットである. インデックスの順序は QFT の後に逆転する. QFT MPO のボンド次元は $\mathcal{R}$ によらず機械精度でも最大 10–20 である.

演習: 指数関数の QTT 表現

$f(x) = e^x$ は区間 $x \in [0, 1]$ に対してボンド次元 1 の QTT 表現を持つことを示せ.

演習: 単位行列の QTT 表現

$f(x, y) = \delta_{xy}$ のボンド次元 1 の QTT 表現を見つけよ. ここで $x, y = 0, 1, \cdots, 2^{\mathcal{R}} - 1$ である.

Bibliography

[1] U. Schollwock, The density-matrix renormalization group in the age of matrix product states, Ann. Phys. **326**, 96 (2011).

[2] Y. N. Fernández et al., Learning tensor networks with tensor cross interpolation: new algorithms and libraries, Arxiv Preprint Arxiv:2407.02454 (2024).

[3] I. Oseledets, *Approximation of Matrices with Logarithmic Number of Parameters*, in *Doklady Mathematics*, Vol. 80 (2009), pp. 653–654.

[4] B. N. Khoromskij, O (d log N)-quantics approximation of N-d tensors in high-dimensional numerical modeling, Constructive Approximation **34**, 257 (2011).

[5] J. Chen, E. M. Stoudenmire, and S. R. White, Quantum Fourier Transform Has Small Entanglement, PRX Quantum **4**, 40318 (2023).

[6] H. Shinaoka, M. Wallerberger, Y. Murakami, K. Nogaki, R. Sakurai, P. Werner, and A. Kauch, Multiscale Space-Time Ansatz for Correlation Functions of Quantum Systems Based on Quantics Tensor Trains, Physical Review X **13**, 21015 (2023).