
Higher-Dimensional Problems, Tensor Models and Applications

Renato Pajarola, Rafael Ballester-Ripoll
CP and Tucker: Limitations

- **Scales well** with dimensionality
- Cannot truncate $\lambda$
- Often, too many ranks needed
  - Slower decomposition
  - How to choose $R$?

- **Scales poorly** for $N \geq 4$
- Can truncate core
- Decomposition uses SVD
  - Fast and stable
  - Better control over $R_n$
Tensor Train Decomposition

- More recent model [O10a]
- **One 3D core per dimension**
  - Great for 4D+ dimensions
- Core $n$ has size $R_{n-1} \times I_n \times R_n$
- $R_0 = R_N = 1 \Rightarrow$ **first and last cores are matrices**

3D

4D
Tensor Train Decomposition

- Decomposition time: $O(I^N R)$ (SVD-based)
- Reconstruction time (full): $O(I^N R)$
- Storage space: $O(NIR^2)$
- Reconstruction time (one element): $O(NR^2)$
Tensor Train Manipulation

- **Linear transforms** (DCT, DFT, separable DWT)
- **Convolution**
- Element-wise **product** between two tensors
- Element-wise **function of a tensor**
- **Derivatives** and **integrals**
- N-variate **projections** and **statistics**
- **Adaptive sampling**

All in time $O(NIR^3)$ at most
Tensor Train Reconstruction (4D)

- Element-wise:

\[ \mathcal{A}(i_1, i_2, i_3, i_4) \approx \sum_{r_1, r_2, r_3} G^{(1)}(i_1, r_1) \cdot G^{(2)}(r_1, i_2, r_2) \cdot G^{(3)}(r_2, i_3, r_3) \cdot G^{(4)}(r_3, i_4) \]

- Full:

\[ \mathcal{A} \approx \sum_{r_1, r_2, r_3} G^{(1)}(\; ; r_1) \circ G^{(2)}(r_1, \; ; r_2) \circ G^{(3)}(r_2, \; ; r_3) \circ G^{(4)}(r_3, \; ;) \]
Computing a Tensor Train (3D)

1. **Input**: $I_1 \times I_2 \times I_3$

2. **Reshape** to $I_1 \times I_2 I_3$

3. **Compress (SVD)**: we get $(I_1 \times R_1)$ and $(R_1 \times I_2 I_3)$
Computing a Tensor Train (3D)

1. **Input**: \((R_1 \times I_2 I_3)\)
2. **Reshape** to \(R_1 I_2 \times I_3\)
3. **Compress (SVD)**: we get \((R_1 I_2 \times R_2)\) and \(R_2 \times I_3\)
Computing a Tensor Train
Tensor Networks

- Graphical way to see tensors
  - A free edge is a dimension
Tensor Networks

- Graphical way to see tensors
  - A free edge is a dimension

Matrix
Tensor Networks

- Graphical way to see tensors
  - A free edge is a dimension

4D tensor
Tensor Networks

- Tensors can **connect** with each other (**tensor contraction**)  
  - Sizes must match!

\[ I_1 \quad A \quad I_2 \]

Matrix times vector
Tensor Networks

- Principal component analysis
  - Compress along 1 dimension

![Diagram of Tensor Networks](image)
Tensor Networks

- Singular value decomposition
  - Compress along 2 dimensions
Tensor Networks

4D Tucker decomposition
Tensor Networks

4D tensor train
Tensor Networks

Tucker + tensor train
Blessing of Dimensionality

• Reshaping data can **reveal patterns and structure**
  ‣ Example: multiresolution analysis in 1D signals
• Curse of dimensionality — *blessing of dimensionality*
  ‣ The more dimensions, the more correlation we can potentially remove
• **TT copes well with dimensionality**
• Strategy: add **new dimensions** to the data (*tensorization*)
Quantized Tensor Train

- Suppose we reshape a vector into a cube
- Similarity between segments $\leftrightarrow$ similarity between tensor slices
Quantized Tensor Train

1. Reshape: $2^N \leftrightarrow 2 \times \ldots \times 2$
2. Compress as ND tensor train
3. Example: 1D vector with 32 elements $\rightarrow$ 5D QTT
Quantized Tensor Train

- Cube to QTT:
  - Reshape: $2^N \times 2^N \times 2^N \leftrightarrow (2 \times 2 \times 2)^N \times \ldots \times (2 \times 2 \times 2)$
- Each core slice maps to half the voxels
Quantized Tensor Train

- Core 1 maps to:
Quantized Tensor Train

- Core 2 maps to:
Quantized Tensor Train

- Core 3 maps to:
Quantized Tensor Train

- Core 4 maps to:
Quantized Tensor Train

- Core 5 maps to:
Quantized Tensor Train

- Core 6 maps to:
Quantized Tensor Train

- Core 7 maps to:
Quantized Tensor Train

- Core 8 maps to:
Quantized Tensor Train

- Core 9 maps to:
Quantized Tensor Train

- It is in fact a 3D multiresolution decomposition
- Connections to the wavelet transform [OT10], [K13]
Tensor Decomposition Software

- Many open-source packages exist

- **Tensor format:**
  - CP
  - Tucker
  - TT
  - Variants

- **Data type:**
  - Dense
  - Sparse
  - Missing values

- **Sampling strategy:**
  - Fixed
  - Adaptive

- **Languages:**
  - MATLAB
  - C++
  - FORTRAN
  - Python (more recently)
## Tensor Decomposition Software

<table>
<thead>
<tr>
<th>Link</th>
<th>Paper/manual</th>
<th>Last Update</th>
<th>Language</th>
<th>Tensor Format(s)</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>tensorlib</td>
<td>Documentation</td>
<td>2014</td>
<td>Python</td>
<td>CP; Tucker</td>
<td>Dense compression</td>
</tr>
<tr>
<td>h Tucker</td>
<td>Paper</td>
<td>2013</td>
<td>MATLAB</td>
<td>HT</td>
<td>Dense compression; adaptive sampling</td>
</tr>
<tr>
<td>GeomCG</td>
<td>Paper</td>
<td>2013</td>
<td>MATLAB</td>
<td>Tucker</td>
<td>Tensor completion</td>
</tr>
<tr>
<td>TDALAB</td>
<td>Manual</td>
<td>2012</td>
<td>MATLAB</td>
<td>CP; Tucker</td>
<td>Dense compression</td>
</tr>
<tr>
<td>FSTD</td>
<td>Paper</td>
<td>2010</td>
<td>MATLAB</td>
<td>Tucker</td>
<td>Adaptive sampling</td>
</tr>
<tr>
<td>Cross3D (variant)</td>
<td>Paper</td>
<td>2008</td>
<td>MATLAB</td>
<td>Tucker</td>
<td>Adaptive sampling</td>
</tr>
</tbody>
</table>
# Tensor Decomposition Software

<table>
<thead>
<tr>
<th>Link</th>
<th>Paper/manual</th>
<th>Last Update</th>
<th>Language</th>
<th>Tensor Format(s)</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tensorlab</td>
<td>Documentation</td>
<td>2016</td>
<td>MATLAB</td>
<td>CP; Tucker; BTD; advanced variants</td>
<td>Dense and sparse compression; tensor completion; adaptive sampling</td>
</tr>
<tr>
<td>tucker_opt</td>
<td>Paper</td>
<td>2015</td>
<td>MATLAB</td>
<td>Tucker</td>
<td>Tensor completion</td>
</tr>
<tr>
<td>NTFLib</td>
<td>-</td>
<td>2015</td>
<td>Python</td>
<td>NTF</td>
<td>Sparse compression</td>
</tr>
<tr>
<td>vmmlib</td>
<td>vmmlib classes</td>
<td>2015</td>
<td>C++</td>
<td>CP; Tucker</td>
<td>Dense compression</td>
</tr>
<tr>
<td>scikit-tensor</td>
<td>-</td>
<td>2015</td>
<td>Python</td>
<td>CP; Tucker; RESCAL; DEDICOM; INDSCAL</td>
<td>Dense/sparse compression</td>
</tr>
<tr>
<td>Tensor Toolbox</td>
<td>T. Toolbox classes</td>
<td>2015</td>
<td>MATLAB</td>
<td>CP; Tucker</td>
<td>Dense/sparse compression</td>
</tr>
<tr>
<td>Tensor CUR</td>
<td>Paper</td>
<td>2015</td>
<td>Python</td>
<td>Tucker</td>
<td>Adaptive sampling</td>
</tr>
</tbody>
</table>
Tensor Decomposition Software

<table>
<thead>
<tr>
<th>Link</th>
<th>Paper/manual</th>
<th>Last Update</th>
<th>Language</th>
<th>Tensor Format(s)</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttpy</td>
<td>Paper 1, 2</td>
<td>2016</td>
<td>Python</td>
<td>TT</td>
<td>Dense compression; adaptive sampling</td>
</tr>
<tr>
<td>TT-Toolbox</td>
<td>(see above)</td>
<td>2016</td>
<td>MATLAB</td>
<td>TT</td>
<td>Dense compression; adaptive sampling</td>
</tr>
<tr>
<td>TTTeMPS</td>
<td>PhD Thesis</td>
<td>2016</td>
<td>MATLAB</td>
<td>TT</td>
<td>Tensor completion</td>
</tr>
<tr>
<td>C++ TT</td>
<td>-</td>
<td>2016</td>
<td>C++</td>
<td>TT</td>
<td>Dense compression; adaptive sampling</td>
</tr>
<tr>
<td>Polara</td>
<td>Paper</td>
<td>2016</td>
<td>Python</td>
<td>Tucker</td>
<td>Sparse compression</td>
</tr>
<tr>
<td>vmmlib-tensor</td>
<td>-</td>
<td>2016</td>
<td>C++</td>
<td>Tucker</td>
<td>Dense compression</td>
</tr>
<tr>
<td>SPLATT</td>
<td>Slides</td>
<td>2016</td>
<td>C++; MATLAB</td>
<td>CP</td>
<td>Sparse compression</td>
</tr>
</tbody>
</table>
Tensor Decomposition Software

- Full list available at:

  https://github.com/rballester/tensor_notes/blob/master/implementations.md
Tensors in Photo-realistic Rendering

- Material properties: high-dimensional data
- Reflectance is recorded (or simulated) according to several parameters
  - View position
  - Light position
  - Texture coordinates
  - Wavelength/color channel

Source: UBO2014 database
Bidirectional Reflectance Distribution Functions

- Reflectance for one single point
- **5D tensor**: $\mathcal{A}(\phi_l, \theta_l, \phi_v, \theta_v, \lambda)$
  - Incoming ray: $(\phi_l, \theta_l)$
  - Outgoing ray: $(\phi_v, \theta_v)$
  - Wavelength: $\lambda$
- Imitate “homogenous” materials
  - Same behaviour at every point
- Types:
  - **Isotropic**: $\mathcal{A}$ independent from $(\theta_l, \theta_v) \rightarrow$ 3D tensor
  - **Anisotropic**: general case

Source: [RSK12]
Bidirectional Texture Functions

- Reflectance for every 2D material point
- **7D tensor**: $\mathcal{A}(x, y, \phi_l, \theta_l, \phi_v, \theta_v, \lambda)$
  - Incidence point: $(x, y)$
  - Incoming ray: $(\phi_l, \theta_l)$
  - Outgoing ray: $(\phi_v, \theta_v)$
  - Wavelength: $\lambda$
3D Reflectance Fields

• Light transport in a 3D scene
  ‣ Reflection between every possible pair of points

• 11D tensor: \( A(x_1, y_1, z_1, \phi_1, \theta_1, x_2, y_2, z_2, \phi_2, \theta_2, \lambda) \)
  ‣ 3D source point: \((x_1, y_1, z_1)\)
  ‣ Incoming ray: \((\phi_1, \theta_1)\)
  ‣ 3D reflection point: \((x_2, y_2, z_2)\)
  ‣ Outgoing ray: \((\phi_2, \theta_2)\)
  ‣ Wavelength: \(\lambda\)
Dimensionality Reduction

- **Dimensions** are sometimes **merged together**
  - Depending on the sampling strategy
- Example: UBO2003 BTF materials
  - Light: $1$ index $l$ instead of $(\phi_l, \theta_l)$
  - View: $1$ index $v$ instead of $(\phi_v, \theta_v)$
  - 5D **tensor** instead of 7D tensor
- **Unmerging would require resampling**
  - Introduces additional error
  - Higher sampling density around the pole
    - More redundancy
Sparse BTF Compression

- **K-SVD for many dimensions**

  - $A \approx DX$ where:
    - $D$ is a dictionary matrix (dense)
    - $X$ is the translation matrix (sparse)

- For tensors: express **each slice as a sum of few slices** (words/atoms)

Source: [RK09]
Sparse BTF Compression

• [RK09]: compress the dictionary iteratively for all modes
  ‣ Result:
  \[
  \mathcal{A}(i_1, \ldots, i_N) \approx \sum_{r_1, \ldots, r_{N-1}} \mathcal{D}(i_1, r_1) \cdot \mathcal{K}^{-1}(r_1, i_2, r_2) \cdot \ldots \cdot \mathcal{K}^{(N-1)}(r_{N-1}, i_N)
  \]
  ‣ \(\mathcal{D}\) is dense, \(\mathcal{K}^{-n}\) are sparse
• An (early) \textbf{sparse version} of the TT

• \textbf{Good compression rate}

• Disadvantages:
  ‣ K-SVD is expensive
  ‣ How to optimize parameters \(R_1, \ldots, R_{N-1}\)?
  ‣ Sparsity \(\rightarrow\) reconstruction hard to parallelize
TT BTF Compression

- [BP16]: compress the BTF into a **dense TT** using the **TT-SVD algorithm**
  - SVD algorithm is **fast**
  - Adaptive truncation → **ranks** $R_1, \ldots, R_{N-1}$ **automatically found**
- Dimension order is important!
  - Put **smallest dimension** (color) in the **center**
TT BTF Compression

- Reconstruction: dense vector-times-matrix
  - Good parallelization
  - Also during interpolation
- Optionally: Tucker + TT
Clustered Tensor Approximation

- Idea [TS06], [TS12], [T15]:
  - **Cluster** parts of the tensor together. Then, **compress** each cluster independently.
- Example [TS12]: cluster slices together
  \[ \mathcal{A}(:, \ldots, J_n, \ldots, :) \text{ with } J_n \subset \{1, \ldots, I_n\} \]
- Example [T15]: cluster sub-tensors together
  \[ \mathcal{A}(J_1, \ldots, J_N) \text{ with } J_1 \subset \{1, \ldots, I_1\}, \ldots, J_N \subset \{1, \ldots, I_N\} \]

Source: [T15]
Clustered Tensor Approximation

- **Generalization of sparse Tucker and CP**
  - Instead of summing rank-1 components, we sum small $R_1 \times \ldots \times R_N$ cores
- **Fast reconstruction**

Source: [T15]
Recap

• We often have to deal with **many dimensions**
  ‣ Complex structures in e.g. graphics
  ‣ **Tensorization** → dimensionality can be good!

• Typical requirements:
  ‣ **Good compression rates**
  ‣ **Reasonable compression time**
  ‣ **Fast decompression**

• There are tensor models specifically tailored for these needs
  ‣ **TT (tensor train)**, **QTT** (quantized tensor train)
  ‣ **Sparse** tensor decompositions
  ‣ **Clustered** tensor decompositions

Completion, Inpainting and Adaptive Sampling

Renato Pajarola, Rafael Ballester-Ripoll
Incomplete Data

• Only some signal parts are available
  ‣ Limited observations (e.g. scatter points):
    ‣ Sensor was damaged
    ‣ Or: tensor too big to sample completely!

• We want to **predict** the rest

• Related keywords:
  ‣ Interpolation
  ‣ Completion
  ‣ Recovery
Incomplete Data

• Only *some* signal parts are available
  ‣ **Limited observations** (e.g. scatter points):
    ‣ Sensor was damaged
    ‣ Or: tensor too big to sample completely!

• We want to **predict** the rest

• Related keywords:
  ‣ Interpolation
  ‣ Completion
  ‣ Recovery
Related Work

• Many interpolation techniques:
  ‣ ND splines
  ‣ Radial basis functions
  ‣ Gaussian process models (*kriging*)
  ‣ Texture synthesis, e.g. Perlin noise
  ‣ Other predictive models (autoregression, SVM, NN, etc.)

• Frequent assumptions:
  ‣ **Local** spatial correlation
  ‣ **Smoothness**
Tensor Completion

- Interpolation with tensor decomposition
- All samples contribute with the same weight to a prediction
- Assumption: low rank
  - We don’t need smoothness
- Smooth usually means low rank
  - In the sense of “using few frequencies”
  - Filter with a separable kernel that uses $k$ frequencies $\rightarrow$ result has rank $\leq k$
Smoothness vs. Rank

\[ \sigma = 0 \]

\[
\ln(\sigma)
\]

Singular values
Smoothness vs. Rank

\[ \sigma = 2 \]
Smoothness vs. Rank

\[ \sigma = 4 \]
Smoothness vs. Rank

\[ \sigma = 6 \]
Smoothness vs. Rank

\[ \sigma = 8 \]
Smoothness vs. Rank

\[ \sigma = 10 \]
Smoothness vs. Rank

\[ \sigma = 12 \]
Smoothness vs. Rank

\[ \sigma = 14 \]
Smoothness vs. Rank

\[ \sigma = 16 \]
Smoothness vs. Rank

\[ \sigma = 18 \]
Smoothness vs. Rank

- Matrix rank is in fact a good **perceptual metric**
- E.g. [NL10]:
  - Compute SVD from (a) **reference** image; (b) **distorted** image
  - Score: combined similarity of
    - Pairwise **singular vectors**
    - Pairwise **singular values**
  - Tested with Gaussian blur, JPEG compression, quantization noise, etc.
Smoothness vs. Rank

• So: smooth usually means low rank

• But: **low rank ⇒ smooth**
  ‣ We do not require smoothness!
  ‣ E.g. we can permute slices → **rank does not change**
Rank Invariance

Rank $R$
Rank Invariance

Still rank $R$!
Rank Invariance

Still rank $R$!
Rank Invariance

Still rank $R$!
Rank Invariance

Still rank $R$!
Tensor Completion

- $\Omega$: known sample positions
- $A|_\Omega$: known sample values
- We look for $X$ such that:
  - $A|_\Omega - X|_\Omega = 0$; or
  - $\|A|_\Omega - X|_\Omega\|$ is penalized
    - In denoising applications: $\Omega$ is the whole domain
- Many variants exist
  - For CP, Tucker, TT, etc.
  - See e.g. [OST08], [KSV13], [CHL14], [FJ15]
Tensor Completion

- Low-rank assumption:
  - \( \text{rank}(\mathcal{X}) = R \); or
  - \( \text{rank}(\mathcal{X}) \leq R \); or
  - \( \text{rank}(\mathcal{X}) \) is penalized

- But: \( \text{rank}(\mathcal{X}) \) is not convex

- Usual substitute: the *nuclear norm*
  - In 2D: \( \sum_i |\sigma_i| \)
  - Has generalizations to ND
  - It is the tightest convex envelope
Slice Generation

• Approach from [CSS08]:
  › Time-varying data: \( \mathcal{A} \approx \mathcal{B} \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)} \)
  › \( U^{(1)} \) and \( U^{(2)} \) are spatial dimensions, \( U^{(3)} \) is temporal
  › Each factor column maps to a slice of \( \mathcal{B} \)
  › Each factor row maps to a slice of \( \mathcal{A} \)
• Each time frame \( j \) comes from row \( U^{(3)}(j,:) \)
• How to generate a new time frame? **Generate a new row**
• They use a **combination** of previous rows (autoregression)
Volume Completion

- $\Omega$: a volume minus a solid region
Volume Completion

• Approach from [BP16]:
  ‣ Fix a core, optimize for best Tucker factors

\[
\arg\min_{U^{(1)}, U^{(2)}, U^{(3)}} \| A - B \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)} \|
\]
Volume Completion

\[
\arg\min_{U^{(1)}, U^{(2)}, U^{(3)}} \| \mathcal{A} - \mathcal{B} \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)} \|
\]

• Solution: alternating least squares (ALS):

\[
\left\{
\begin{align*}
1. & \quad U^{(1)} = \mathcal{A}^{(1)} (\mathcal{B} \times_2 U^{(2)} \times_3 U^{(3)})^{\dagger}_{(1)} \\
2. & \quad U^{(2)} = \mathcal{A}^{(2)} (\mathcal{B} \times_1 U^{(1)} \times_3 U^{(3)})^{\dagger}_{(2)} \\
3. & \quad U^{(3)} = \mathcal{A}^{(3)} (\mathcal{B} \times_1 U^{(1)} \times_2 U^{(2)})^{\dagger}_{(3)} \\
4. & \quad \text{Repeat}
\end{align*}
\right.
\]
Volume Completion
Texture Synthesis

- Approach from [WXC+08]:
  - **Example-based synthesis** (after [KEB05])
  - Many 3D patches have to be compared

- Key idea:
  \[
  \mathcal{A}_1 \approx \mathcal{B}_1 \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)} \\
  \mathcal{A}_2 \approx \mathcal{B}_2 \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)}
  \]
  \[
  \Rightarrow \quad \| \mathcal{A}_1 - \mathcal{A}_2 \| \approx \| \mathcal{B}_1 - \mathcal{B}_2 \|
  \]

- Compress patches $32 \times 32 \times 32 \rightarrow 5 \times 5 \times 5$

- **Distance between patches** $\approx$ **distance between their cores**

- Up to x200 speed-up
Texture Synthesis

Source: [WXC+08]
Other Data Types

- Synthesis and completion can be applied to other data types
- Restriction: we need **multidimensional Cartesian grids**
- **Parameterization** is fundamental
  - Example: geometry images
Other Data Types

• With a proper parameterization, tensor approximation can be exploited
• Example: mesh rank truncation
Adaptive Sampling

- Tensor size: $I^N$
  - Often: **too big** to sample completely!
- But: we can **choose** where to sample
- Strategy: learn **adaptively** best locations to sample
Adaptive Sampling

- Example strategy: 2D
  1. Choose a **random point**
  2. **Sample** fibers (row + column)
  3. **Update** current decomposition
  4. Guess next **best point**
  5. **Repeat**
CUR Factorization

• From the chosen samples, **recover** the full tensor

\[
A \approx C \cdot U^{-1} \cdot R
\]

• How?  
  
  ‣ **C**: known columns  
  ‣ **R**: known rows  
  ‣ **U**: intersection
CUR Factorization

\[ A \approx C \cdot U^{-1} \cdot R \]

- If \( \text{rank}(A) = R \), \( R \) columns and rows are enough to recover \( A \) exactly.
- If not, a good heuristic is the maxvol principle:
  - Select \( C \) and \( R \) so that \( \det(U) \) is maximal
    - We want the intersection to span the biggest possible subspace.
- Synonyms:
  - Pseudo-skeleton decomposition
  - Column-row factorization
CUR Factorization

- Application in rendering: [HPB07]
- \( A \) is the lights-samples matrix
  - One column per light source
  - One row per scene sample
  - Total light received by samples = sum of \( A \)'s columns
- \( A \) is low-rank → sample just some rows and columns
  - Ad-hoc search strategy (they cluster columns first)
  - But same driving idea
CUR Factorization

reference (13 min)

Source: [HPB07]
CUR Factorization

432 rows, 864 columns, 13.5 s

Source: [HPB07]
CUR Factorization

300 rows, 300 columns, 6.4 s

Source: [HPB07]
CUR Factorization

reference (20 min)

Source: [HPB07]
CUR Factorization

300 rows, 900 columns, 16.9 s

Source: [HPB07]
CUR Factorization

100 rows, 300 columns, 5.6 s

Source: [HPB07]
CUR Factorization

reference (8 min)

Source: [HPB07]
CUR Factorization

300 rows, 900 columns, 7.9 s

Source: [HPB07]
CUR Factorization

100 rows, 100 columns, 1.6 s

Source: [HPB07]
N-Dimensional Sampling

- **Fiber sampling** generalizes to any dimension
Cross Approximation

• **Generalization** of CUR

• E.g. for:
  ‣ Tucker [OST08], [CC10]
  ‣ TT [O10a], [S11]
Cross Approximation

- Example: sampled and recovered 3D tensor
Surrogate Visualization Models

• We have a simulation depending on multiple parameters
  ‣ Function $f : \mathbb{R}^N \rightarrow \mathbb{R}$ or $\mathbb{R}^M$

• We want a model that quickly predicts the result

• Related keywords:
  ‣ Hyperparameter optimization
  ‣ Meta-modeling
  ‣ Response surface models
Surrogate Visualization Models

• It is a form of machine learning
• But **geared towards visualization:**
  ‣ Prediction (reconstruction) must be extremely fast (**interactive rates**)
    ‣ For many points at once
  ‣ Bonus: get global or local statistics on the data
• If we can **choose** where to sample → **adaptive cross approximation**
Surrogate Visualization Models

- How to gain intuition?
- Typical approach:
  - **Trial and error**: move one parameter, fix the rest (e.g. using loops)
  - Very similar to fiber sampling
- Idea: let cross approximation do the work
  1. **Learn** the tensor
  2. **Navigate** interactively
Interactive Tensor Reconstruction

- We want to reconstruct a **subspace from an interpolated space**
  - It consists of predictions
- E.g. in 4D: $\mathcal{A}[:, i_2 = a, :, i_4 = b]$
- How many slices are possible?

$$\begin{cases}
\mathcal{A}[i_1, i_2, :, :] & \mathcal{A}[:, i_2, i_3, :] \\
\mathcal{A}[i_1, :, i_3, :] & \mathcal{A}[:, i_2, :, i_4] \\
\mathcal{A}[i_1, :, :, i_4] & \mathcal{A}[:, :, i_3, i_4]
\end{cases}$$

- Precomputing infeasible
  - Space needed: $\binom{N}{2} \cdot I^N$
Interactive Tensor Reconstruction

• But: very efficient tensor reconstruction [BPP16]

• A subspace of dimension $M$ can be reconstructed in:
  ‣ $O(IMR + NR)$ ops. (CP)
  ‣ $O(IMR + R^N)$ ops. (Tucker)
  ‣ $O(IMR + N R^2)$ ops. (TT)

• 1D and 2D plots: usually a few milliseconds
Interactive Navigation
Interactive Navigation
Interactive Navigation
Projections

• N-variate projections are very **efficient to compute**
• Idea: sum along all cores except a few
• Example: **projection matrix**
  ‣ Billions of tensor entries
  ‣ Computed via vector-vector and matrix-matrix operations
  ‣ Under 10 milliseconds
From TT to Parallel Coordinates

- Parallel coordinates:
  - A point \((i_1, \ldots, i_N)\) is represented by a polyline \((0, i_1) \to (1, i_2) \to \ldots \to (N-1, i_N)\)
- One vertical bar per dimension
- Example: point \((1, 3, 2, 4)\)
From TT to Parallel Coordinates

• One can define a **dense version** [HW13]
• For every point \((i_1, ..., i_N)\), a polyline is drawn
  ‣ **Opacity** proportional to the **tensor value** \(\mathcal{A}(i_1, ..., i_N)\)
• We assume **linearity**:
  ‣ Opacity of two coincident segments = sum of their opacities
• Every segment \((i_n, j_n) \rightarrow (i_{n+1}, j_{n+1})\) is drawn
  ‣ Opacity: \(\sum \mathcal{A}\) restricted to \(i_n = j_n, i_{n+1} = j_{n+1}\)
• Compute **projection** \(\rightarrow\) very fast from a compressed tensor
Algorithm 1 Plot a parallel coordinates diagram from a TT

Require: $\mathcal{A}$ is a low-rank TT approximation of the simulation $s$

$\mathcal{B} := \text{empty } N\text{-dimensional TT-tensor}$

for $n$ in $1, \ldots, N$ do

$\mathcal{B}(n) := \sum_{i=1}^{I_n} \mathcal{A}(n)(i,:,:)$

end for

$\triangleright \mathcal{B}$ has size $1 \times \ldots \times 1$

for $n$ in $1, \ldots, N - 1$ do

$\mathcal{C} := \mathcal{B}$

$\mathcal{C}(n) := \mathcal{A}(n)$

$\mathcal{C}(n+1) := \mathcal{A}(n+1)$ \hspace{1cm} $\triangleright \mathcal{C}$ has now size $1 \times \ldots \times I_n \times I_{n+1} \times \ldots \times 1$

$S := \text{decompress}(\mathcal{C})$ \hspace{1cm} $\triangleright S$ has now size $I_n \times I_{n+1}$

for $i$ in $1, \ldots, I_n$ do

for $j$ in $1, \ldots, I_{n+1}$ do

source := $(n, i)$

target := $(n+1, j)$

opacity := $S(i, j)$

drawSegment(source, target, opacity)

end for

end for

end for
From TT to Parallel Coordinates
Recap

- **Tensors for visualization:** fast reconstruction
- **Cheap operations** in the **compressed domain**
  - Projections, statistical moments
  - Multiresolution
- **Methods for sampling from full data**
- **Methods for sampling and completing sparse data**
  - Fixed sampling
  - Adaptive sampling