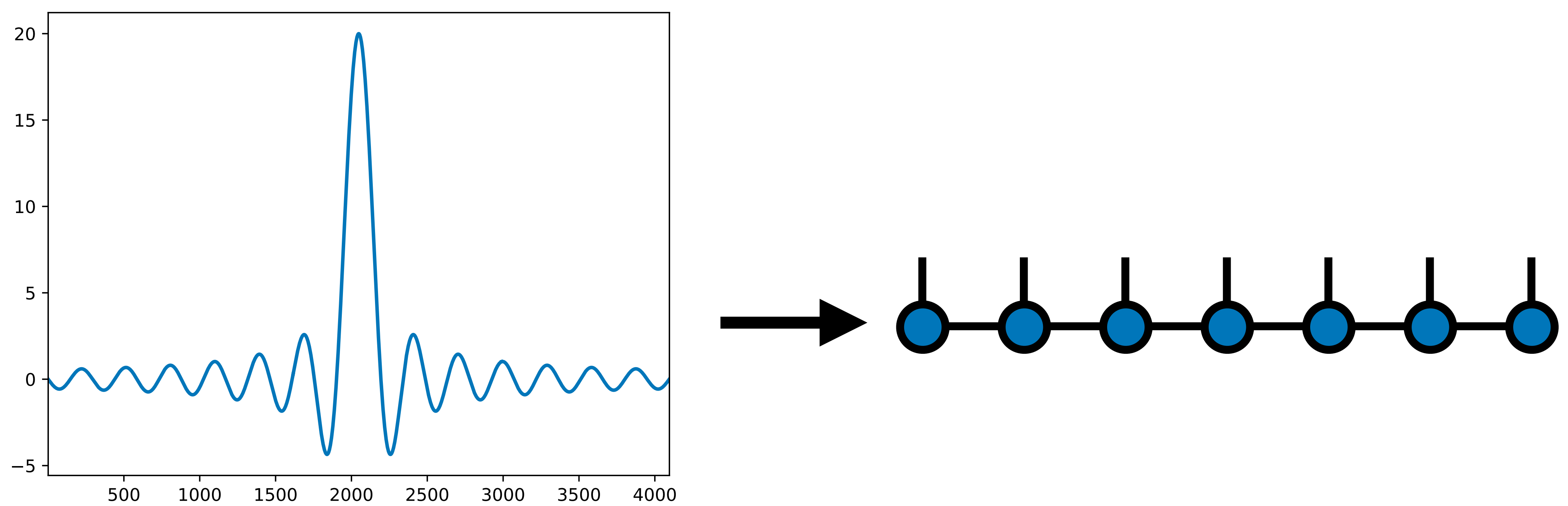
Compressing Functions with Tensor Networks: Applications to PDEs and DFTs



Matt Fishman

Miles Stoudenmire

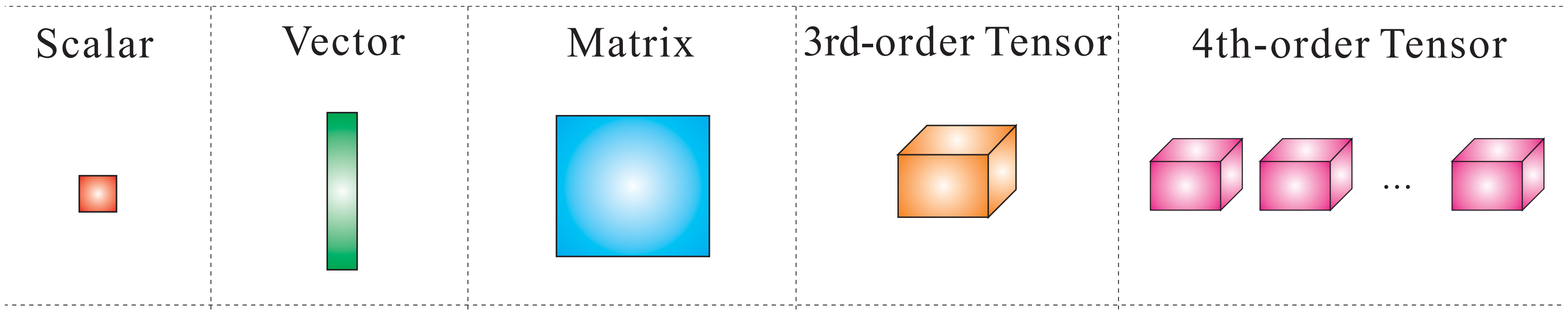
Oct 2022 - FWAM

FLATIRON INSTITUTE

SIMONS FOUNDATION

Tensors are *multi-dimensional arrays or linear maps*

Generalization of vector or matrix



Tensors naturally occur in:

- high-dimensional problems
- <u>continuum</u> problems

Figure credit: Cichocki et al., "Low-Rank Tensor Networks...", arxiv:1609.00893

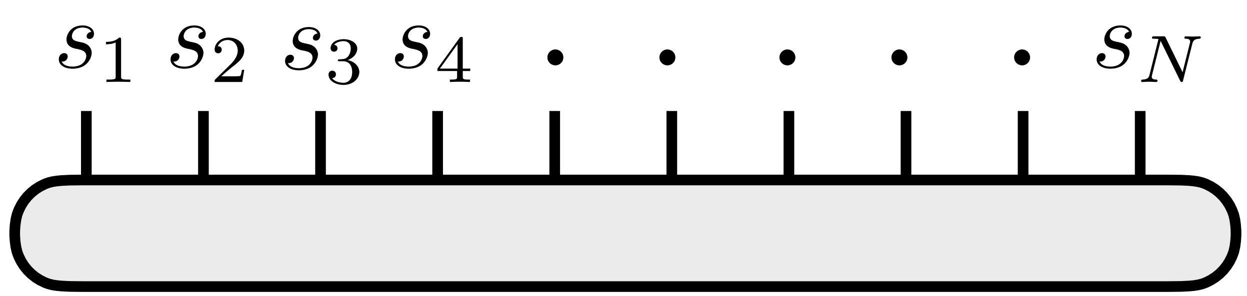
For high-order tensors, one encounters *curse of dimensionality*

$$\underbrace{T_{n_1 n_2 n_3 n_4 n_5 n_6}}_{10^6 \text{ entries}} \qquad n_j = 1, 2, ..., 10$$

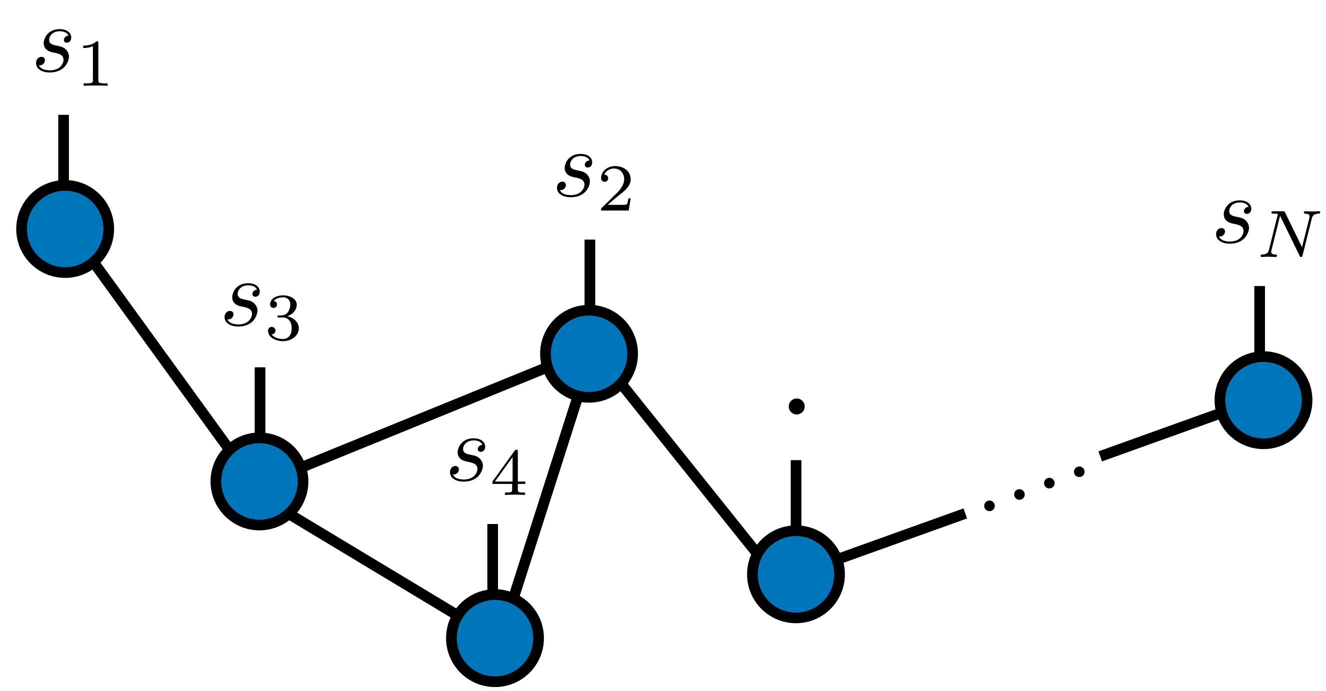
N-th order tensor is *exponential* in N

Tensor networks give a way to break the curse

$$T^{s_1 s_2 s_3 \cdots s_N} =$$



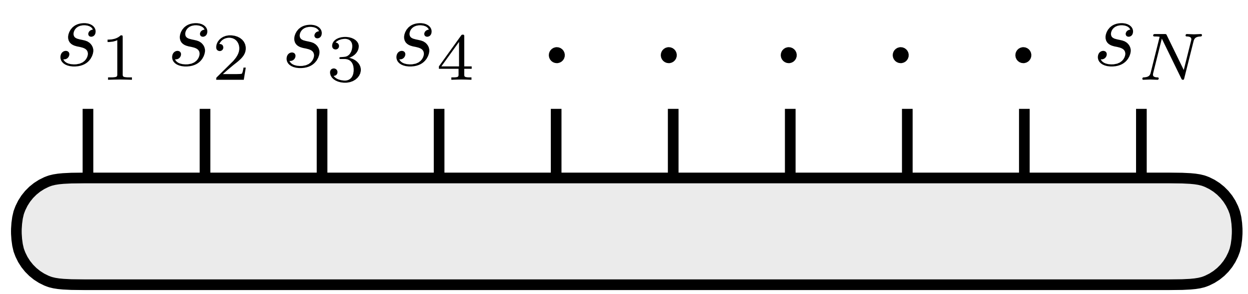
$$T^{s_1 s_2 s_3 \cdots s_N} =$$



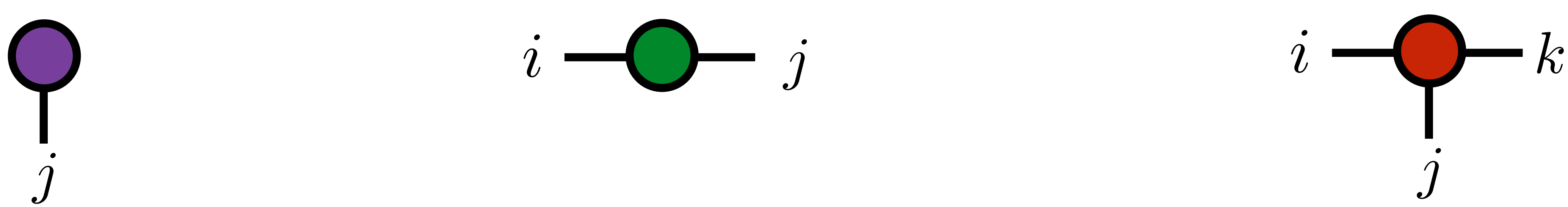
Recall: tensor diagram notation

N-index tensor = shape with N lines

$T^{s_1 s_2 s_3 \cdots s_N} =$

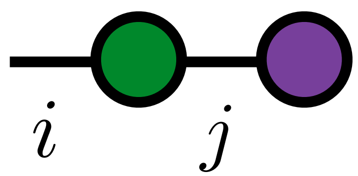


Low-order tensor examples:

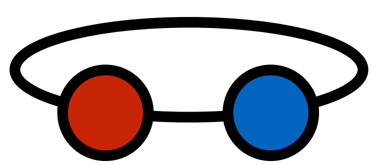


v_j M_{ij} T_{ijk}

Joining lines implies contraction, can omit names



$$\longleftrightarrow \quad \sum_j M_{ij} v_j$$

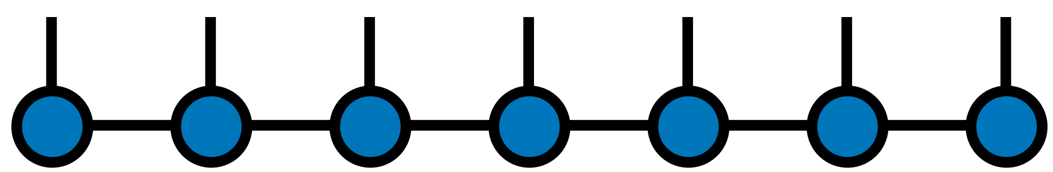


$$\longleftrightarrow \quad A_{ij} B_{ji} \quad = \mathrm{Tr}[AB]$$

Conventionally (since ~1992*), tensor networks used to compress quantum wavefunctions Ψ

$$i\frac{\partial}{\partial t}\Psi(\{\mathbf{x}\},t) = H\Psi(\{\mathbf{x}\},t)$$

Schrödinger equation

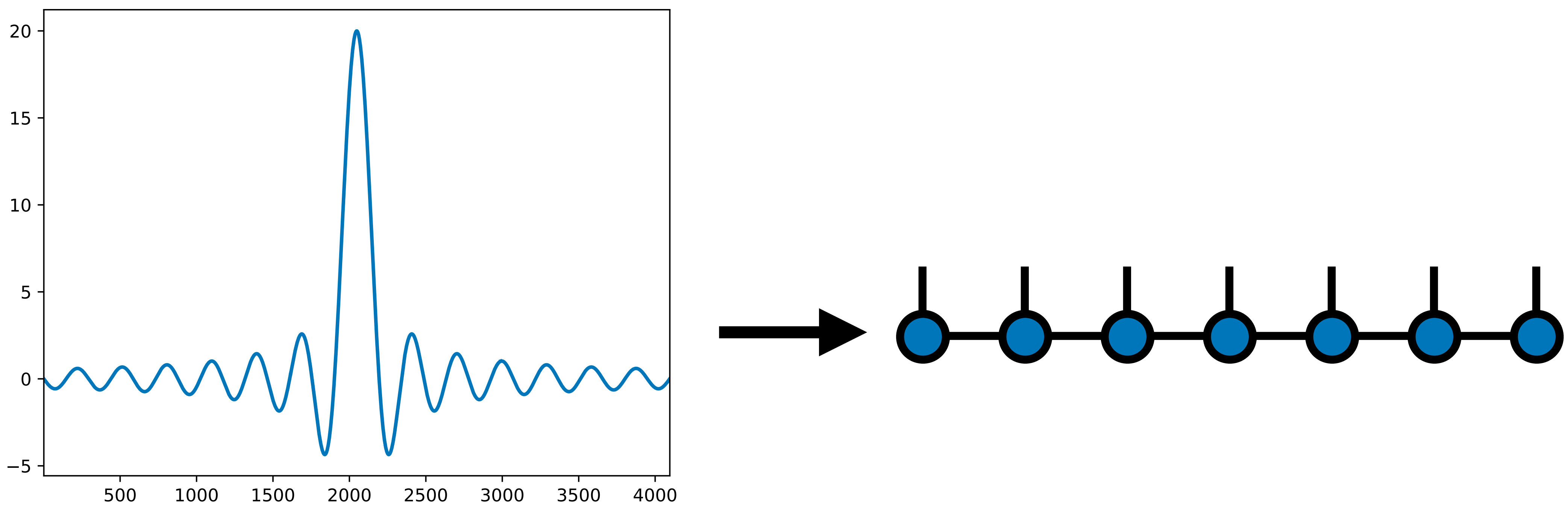
$\Psi =$ 

Primary use case at CCQ

*S.R. White, Phys. Rev. Lett. 69, 2863 (1992)

S. Östlund, S. Rommer, Phys. Rev. Lett. 75, 3537 (1995)

In a parallel development, matrix product state (MPS) (a.k.a. "tensor train") networks can represent *low-dimensional, continuous functions* in compressed form

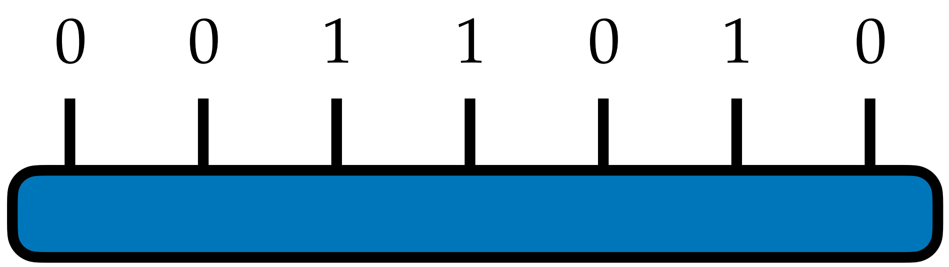


Technique known as "quantized tensor train" (QTT)

B. Khoromskij, Constructive Approximation 34, 257 (2011)
S. Dolgov, B. Khoromskij, D. Savostyanov, J. Fourier Anal. App. 18, 915 (2012)
M. Lubasch, P. Moinier, D. Jaksch, J. Comp. Phys. 372, 587-602 (2018)

How does it work?

Tensor = collection of numbers
labeled by indices



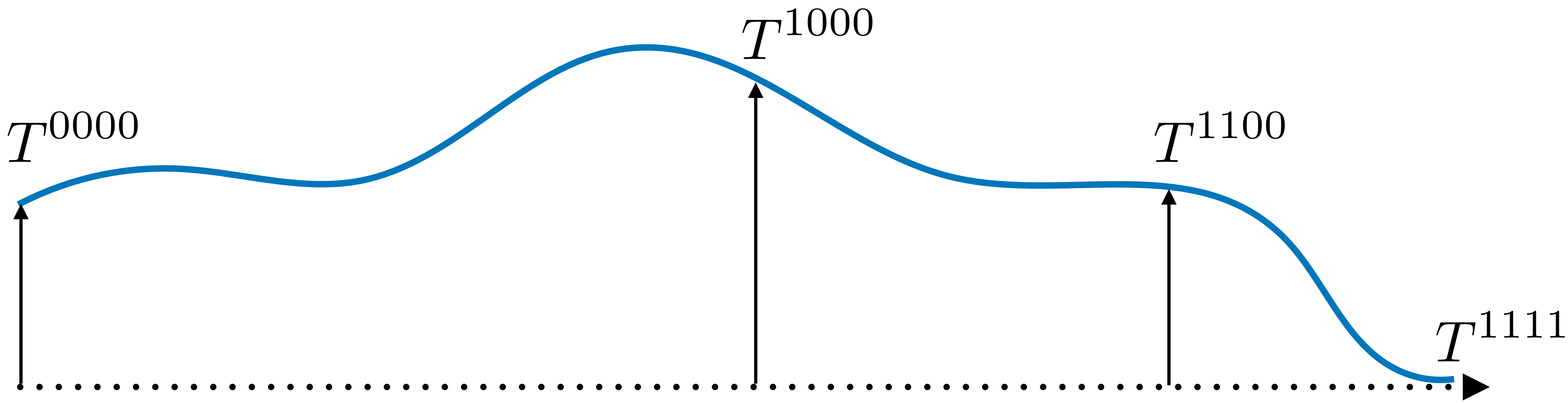
$$= T^{0011010}$$

Interpret indices as binary digits

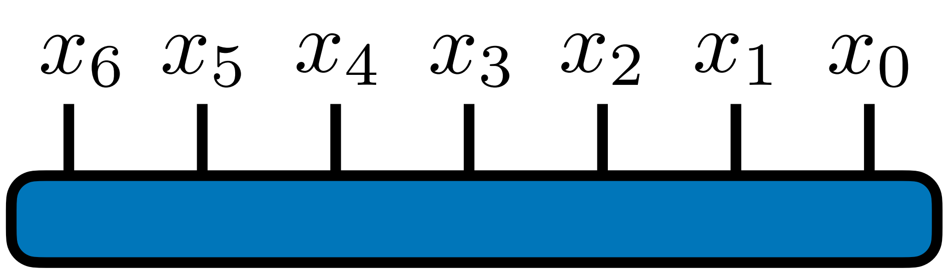
$$\begin{aligned} 0011010 &= 0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \\ &= 26 \end{aligned}$$

Arbitrary function as a tensor

- Binary number (index values) label grid points
- Tensor element = function value

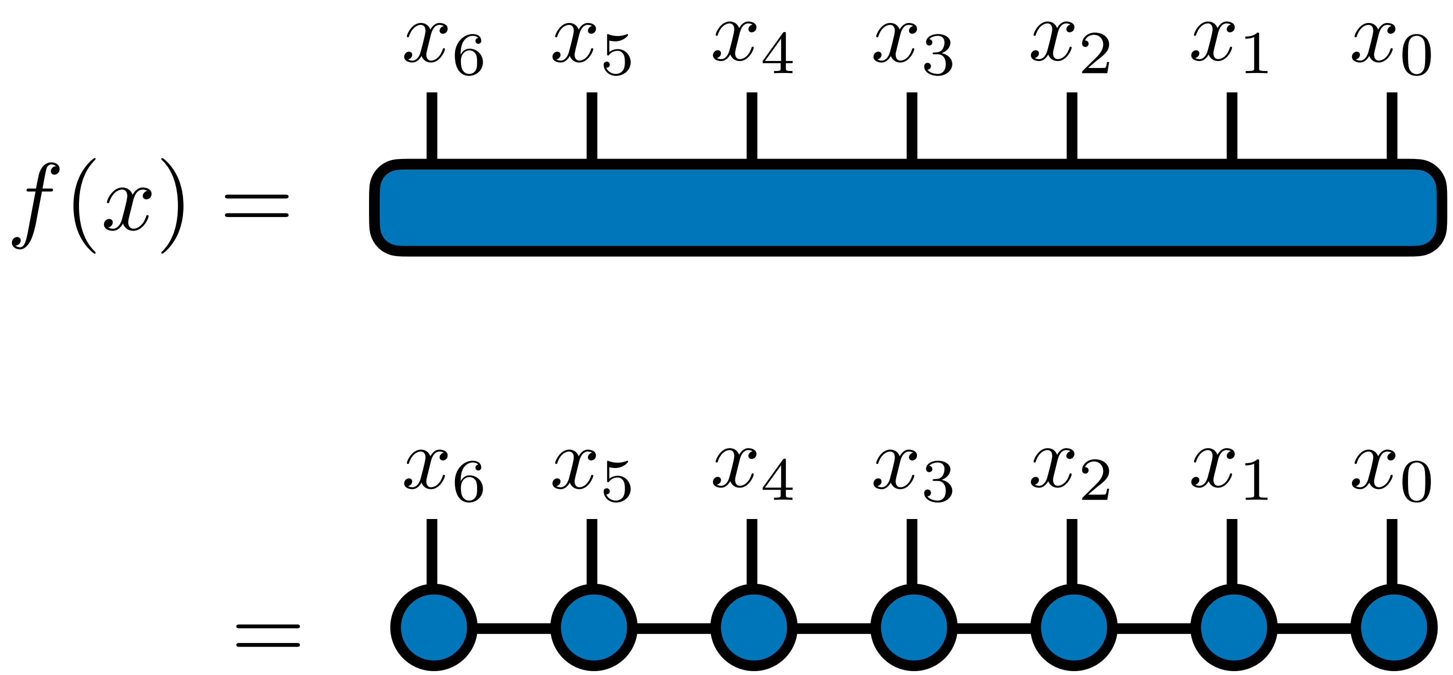


$f(x) =$

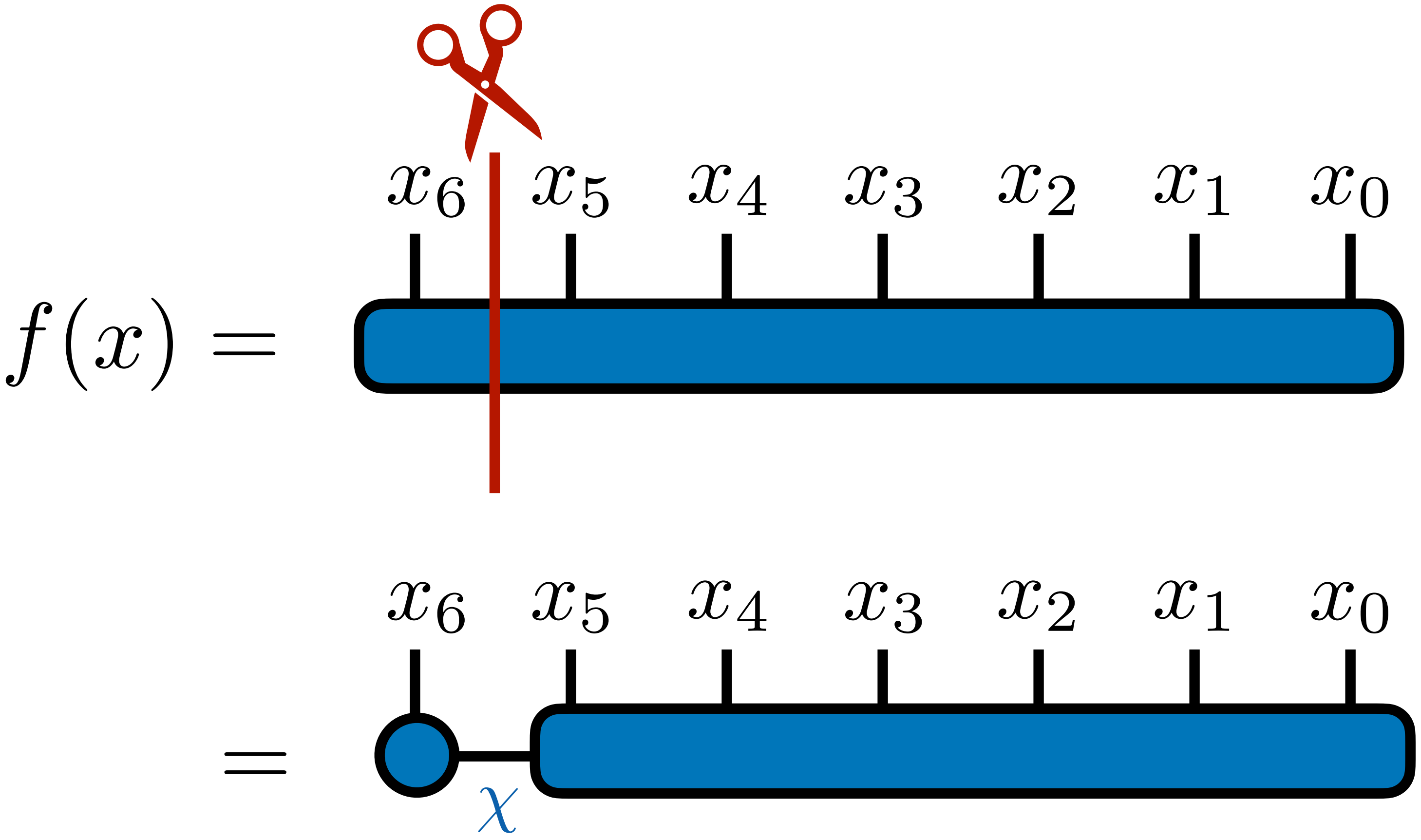


$= T^{x_6 x_5 x_4 x_3 x_2 x_1 x_0}$

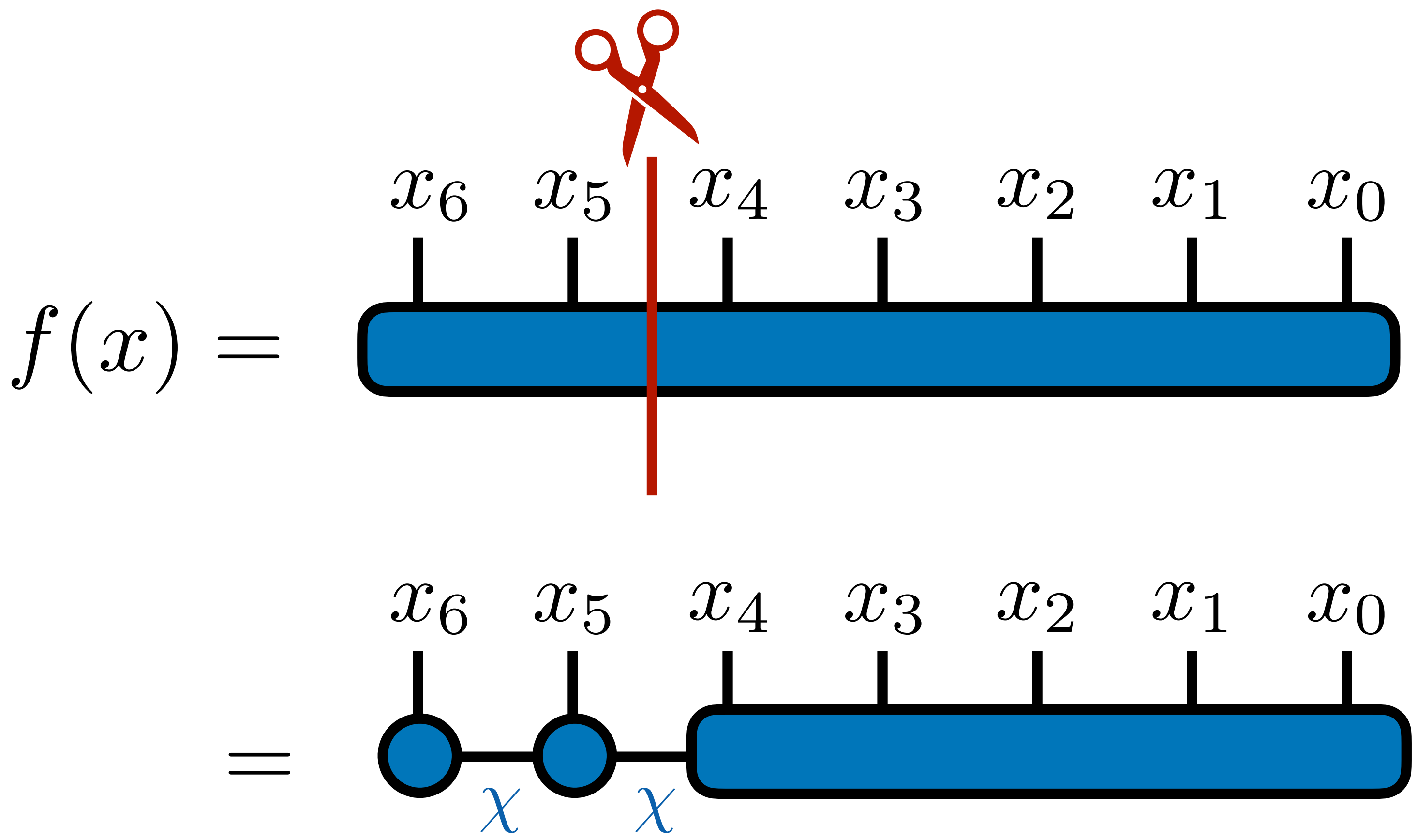
Factorize this "function tensor"
as MPS tensor network



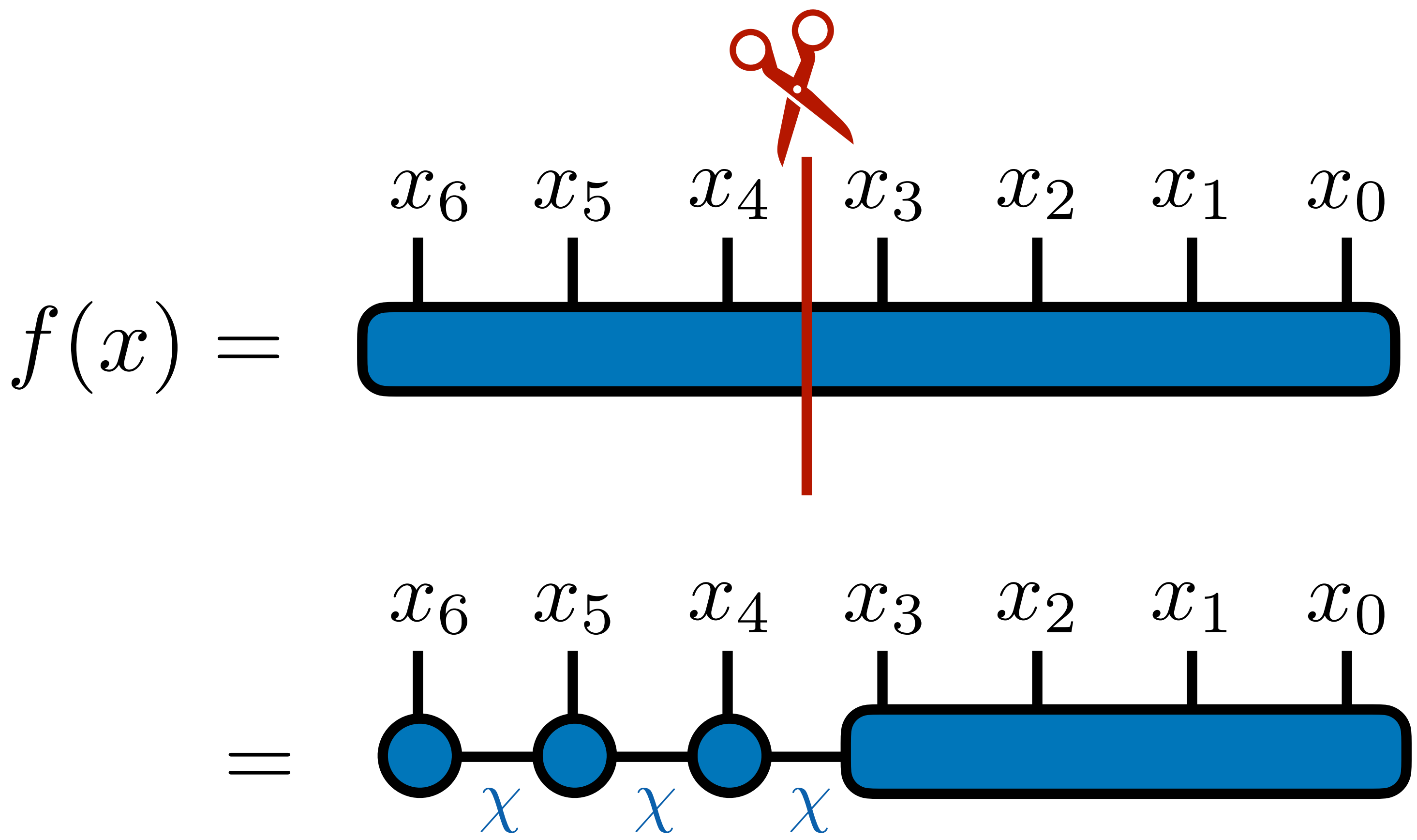
MPS is iterated low-rank decomposition



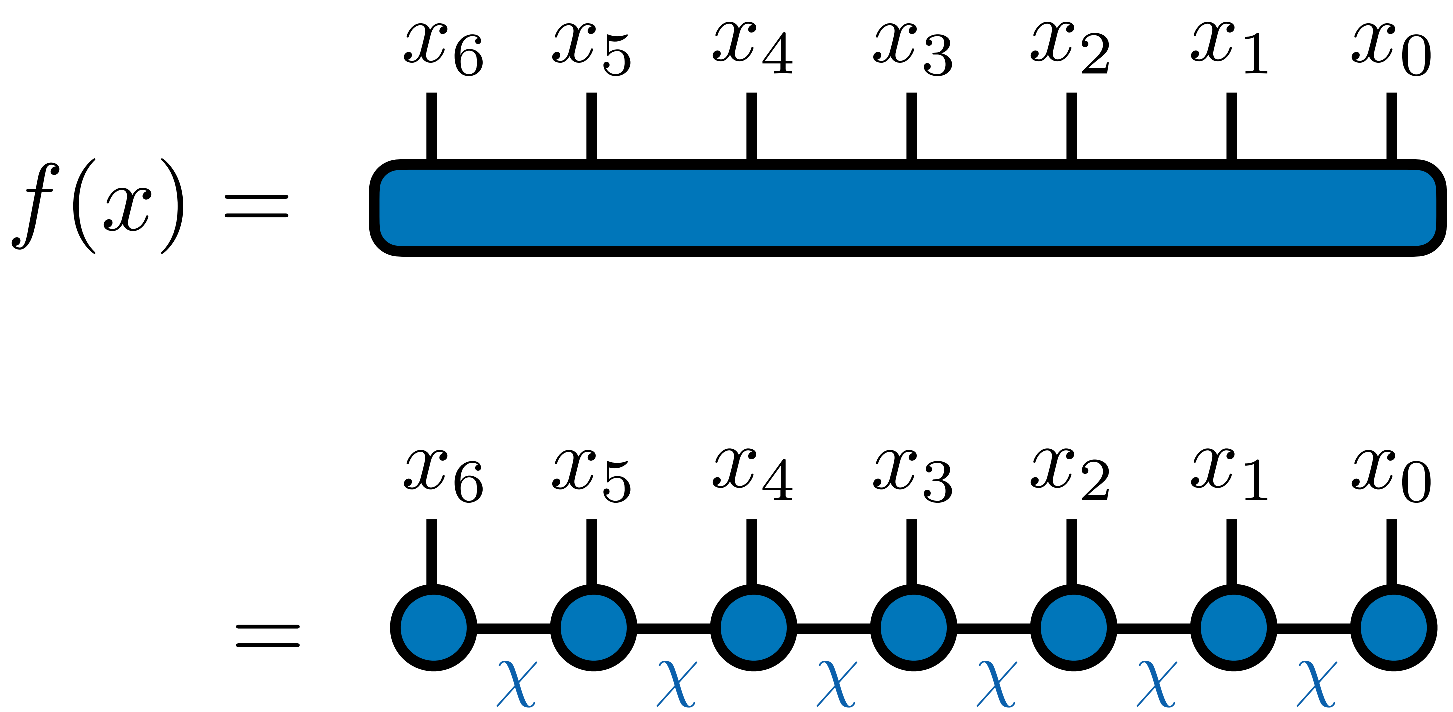
MPS is iterated low-rank decomposition



MPS is iterated low-rank decomposition

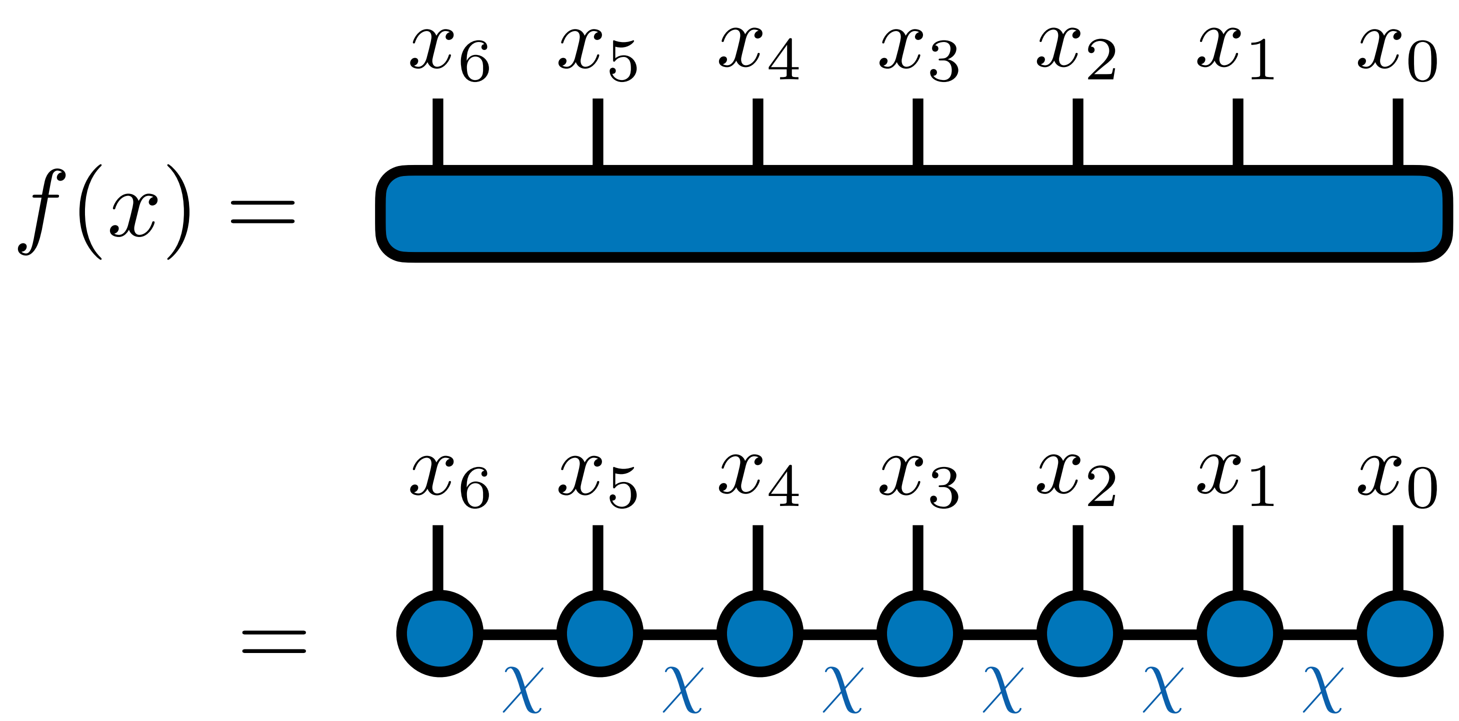


MPS is iterated low-rank decomposition



Obtain computational advantage if ranks χ
("bond dimensions")
can be chosen small without much error

Compression of parameters



Uncompressed tensor = 2^n parameters = # grid points

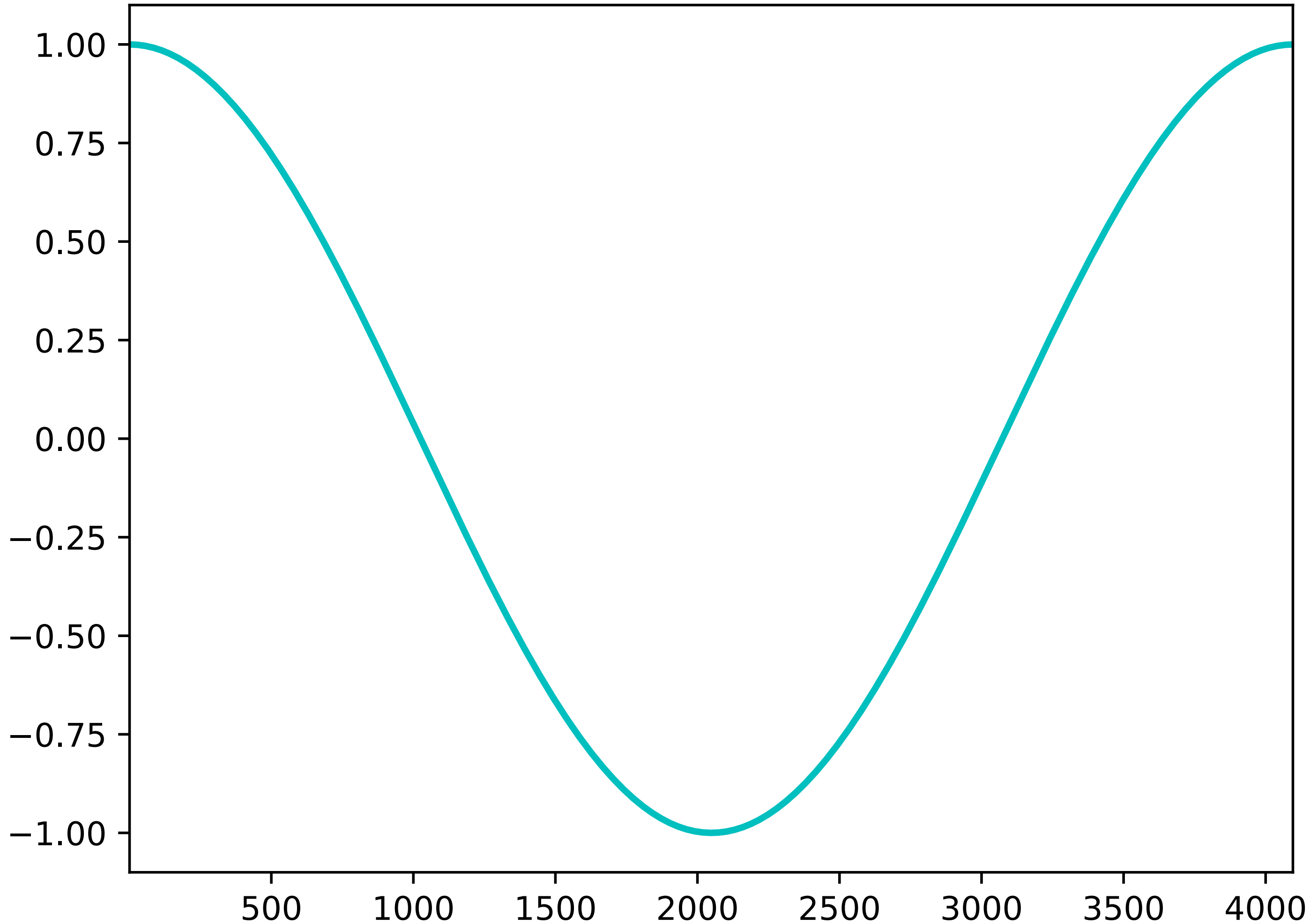
Compressed = $2n\chi^2$ parameters $\ll 2^n$

Evaluating function values, performing integrals scales as $n\chi^2$

Optimization algorithms scale as $n\chi^3$

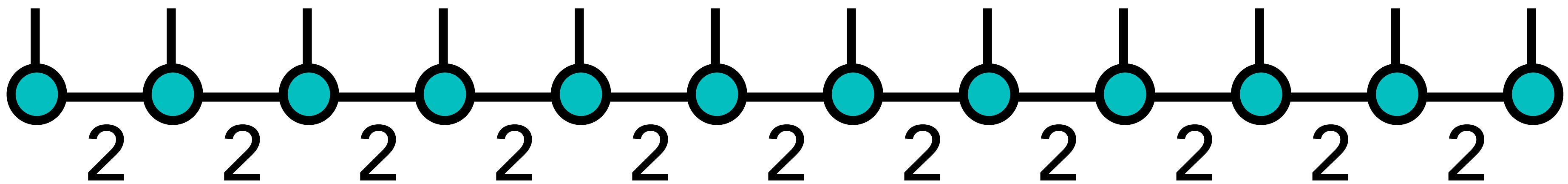
Example Function: Single Cosine

$$||\tilde{f} - f|| = 4.4 \times 10^{-14}$$
$$\max(\tilde{f} - f) = 2.6 \times 10^{-15}$$



$$\cos\left[x - \frac{1}{2}\right]$$

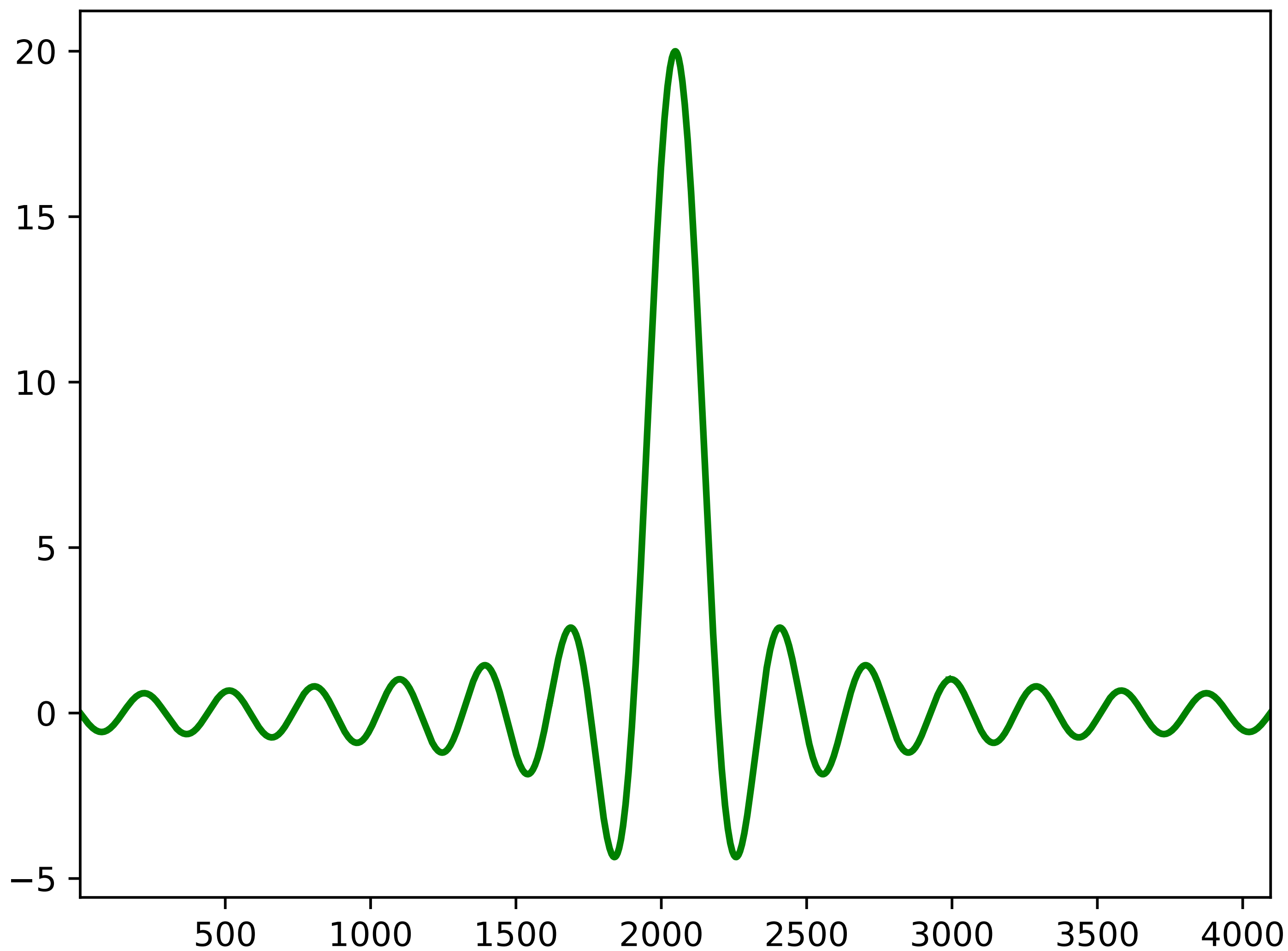
Ranks (n=12):



Example Function: Sum of 20 Cosines

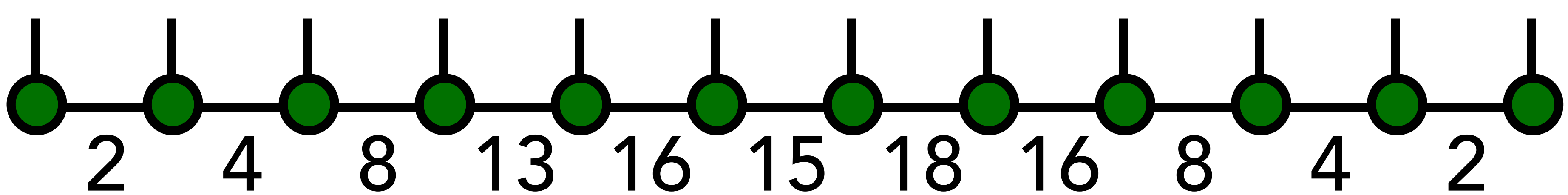
$$||\tilde{f} - f|| = 7.4 \times 10^{-13}$$

$$\max\left(\tilde{f} - f\right) = 1.2 \times 10^{-13}$$



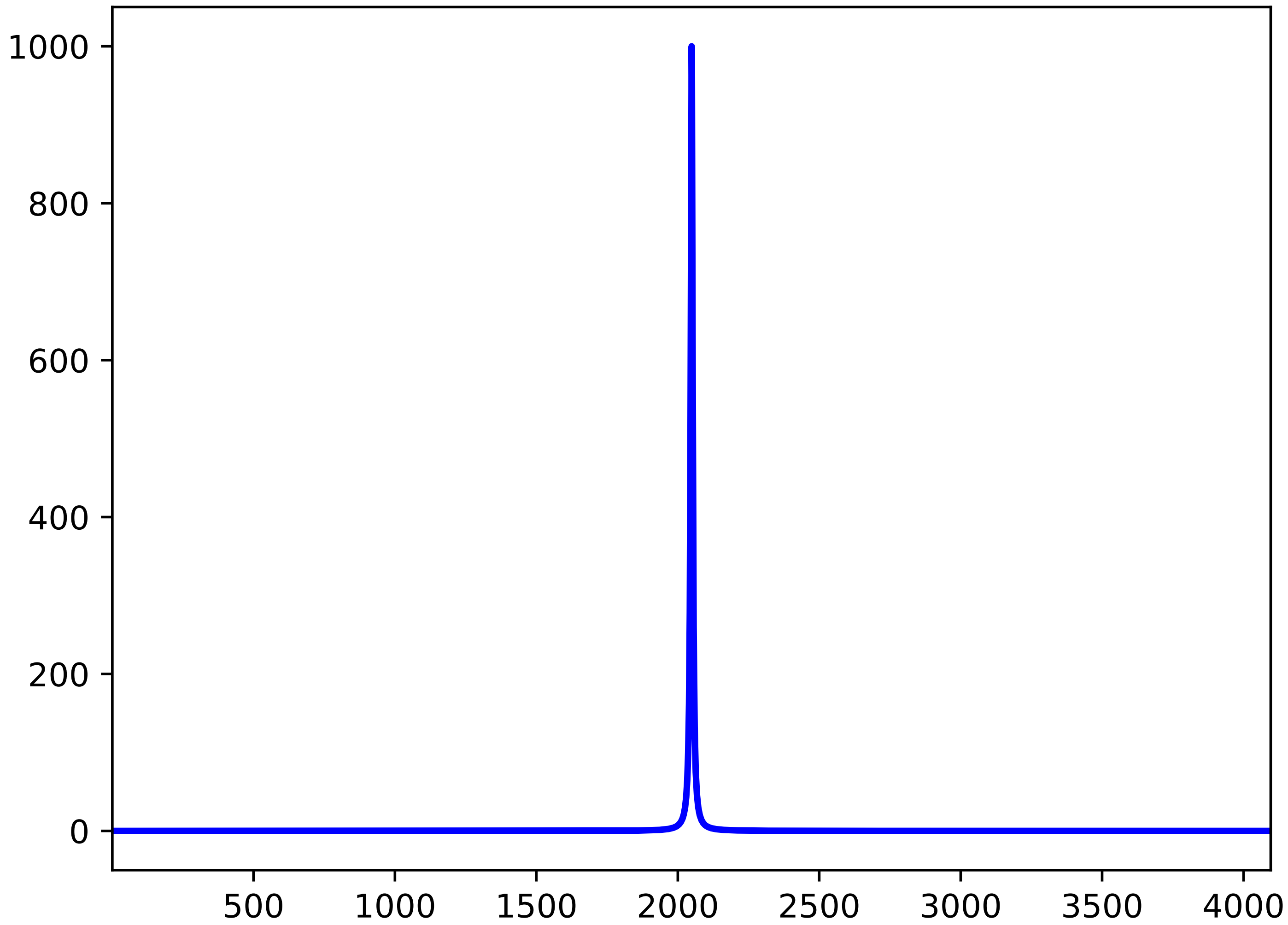
$$\sum_{j=1}^{20} \cos\left[1.1 \cdot (4j-2) \cdot (x - \frac{1}{2})\right]$$

Ranks (n=12):



Example Function: Lorentzian

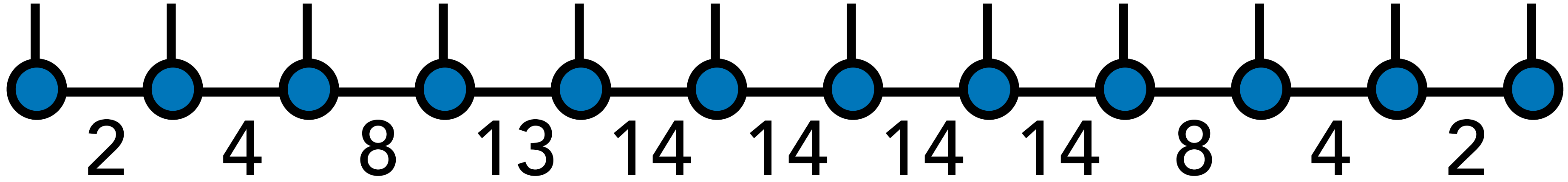
$$||\tilde{f} - f|| = 1.5 \times 10^{-11}$$
$$\max(\tilde{f} - f) = 3.9 \times 10^{-12}$$



$$\frac{a}{(x - \frac{1}{2})^2 + a^2}$$

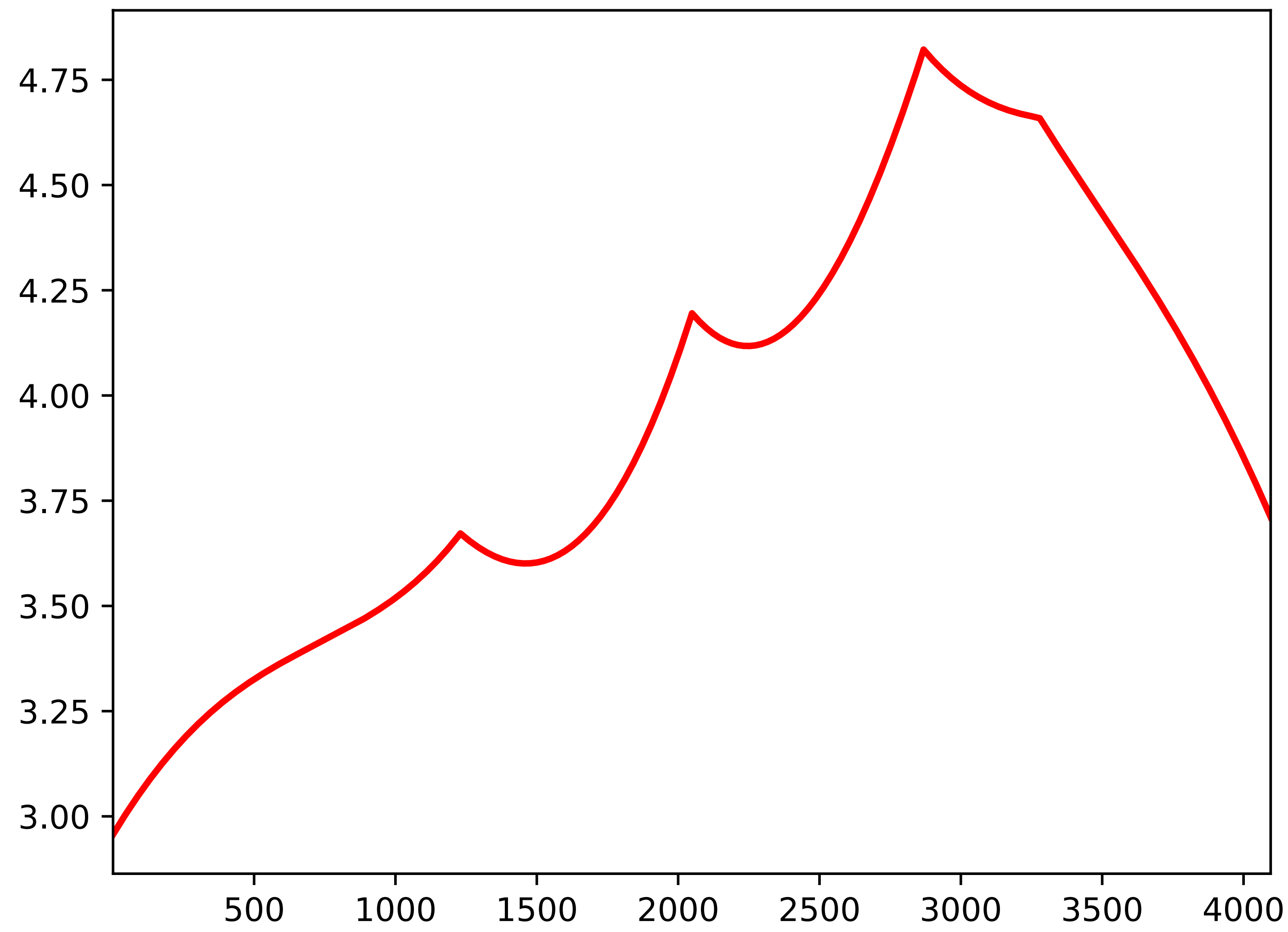
$$a = 0.001$$

Ranks (n=12):



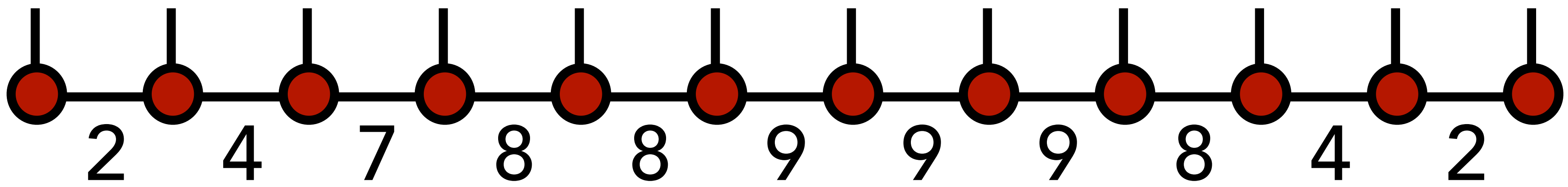
Example Function: Cosine Plus Cusps

$$||\tilde{f} - f|| = 3.1 \times 10^{-12}$$
$$\max(\tilde{f} - f) = 2.5 \times 10^{-13}$$

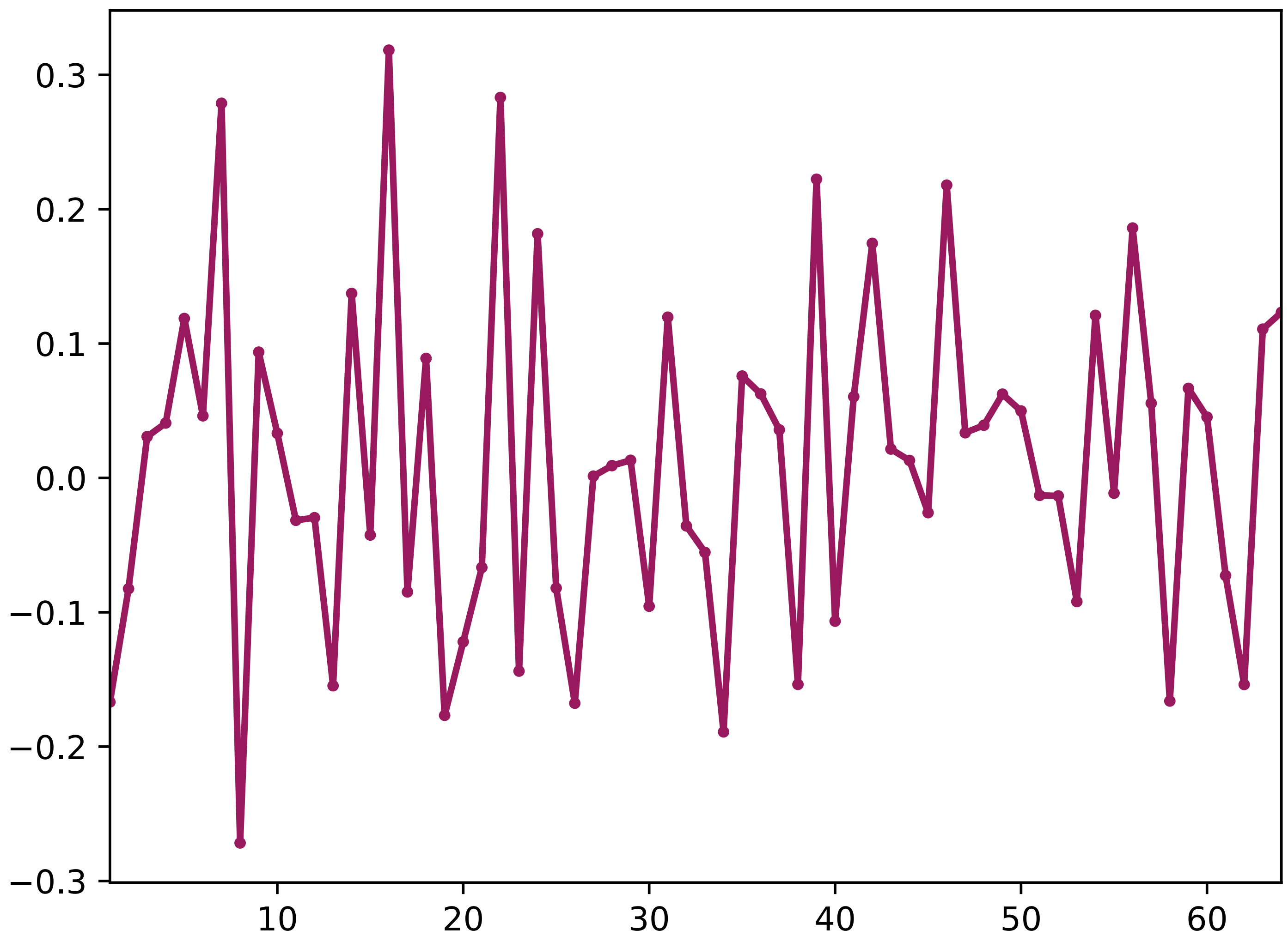


$$\cos(2\pi x) + e^{-3|x-0.3|} + 3e^{-2|x-0.5|} + 2e^{-3|x-0.7|} + e^{-2|x-0.8|}$$

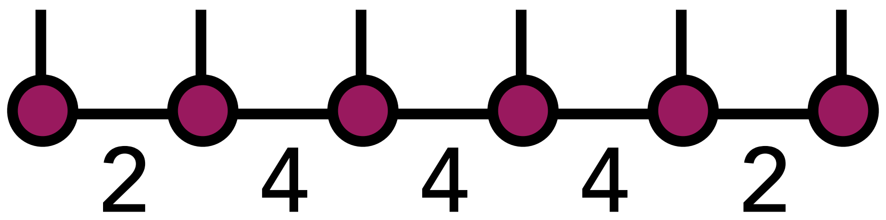
Ranks (n=12):



Example Function: Random MPS $\chi = 4$

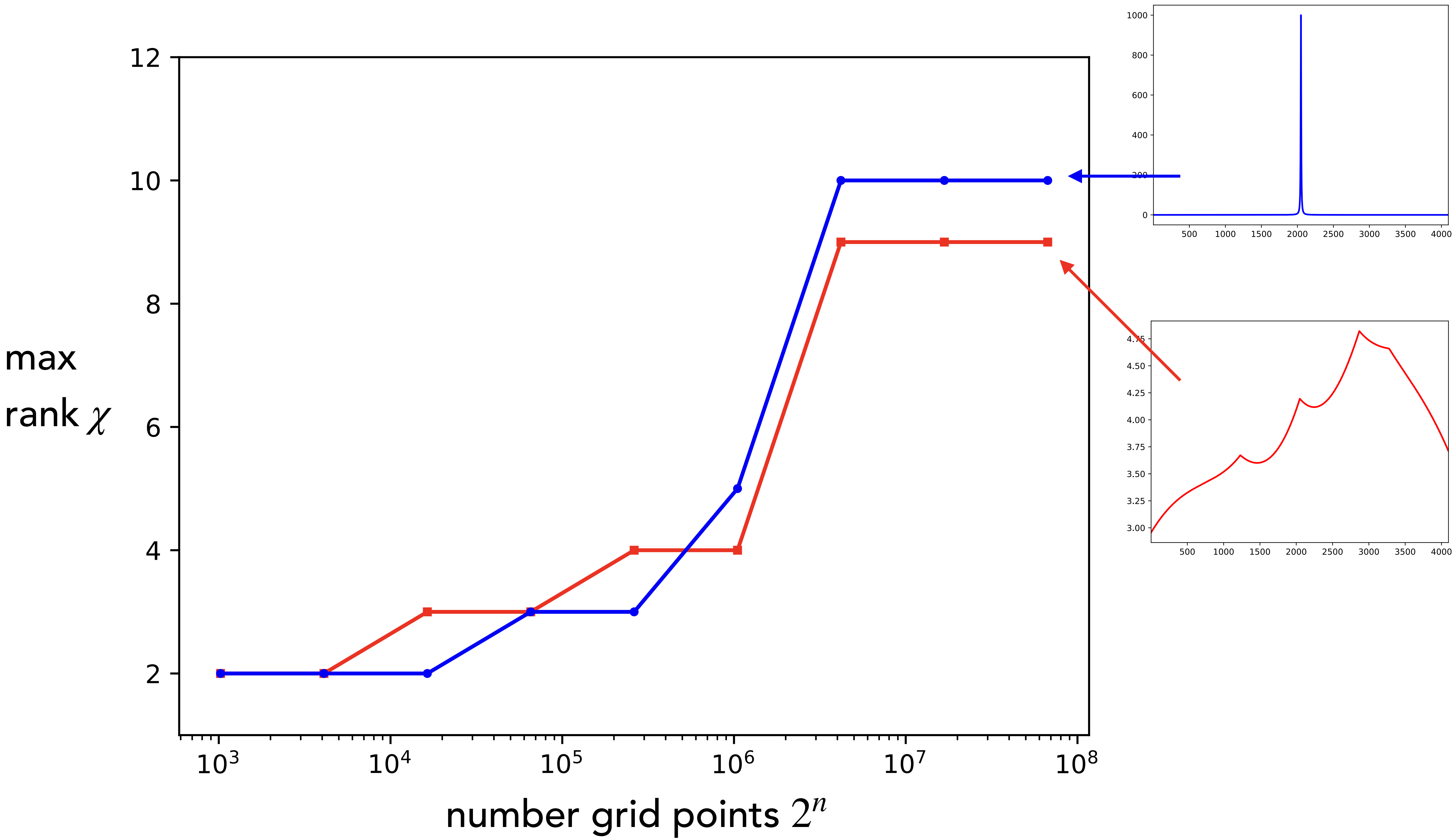


Ranks (n=6):



MPS Rank Versus Grid Size

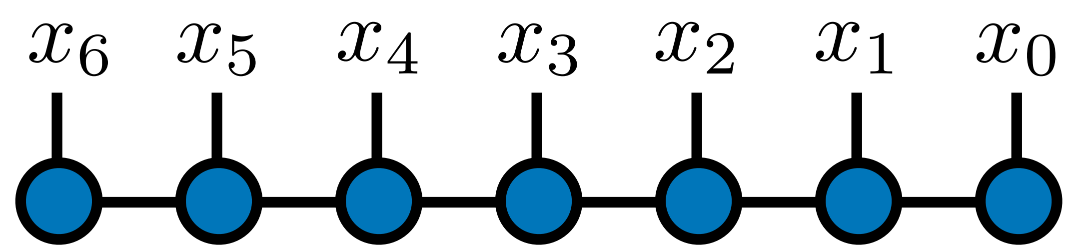
Using SVD threshold $\epsilon = 10^{-10}$



Very different use of MPS versus wavefunction:

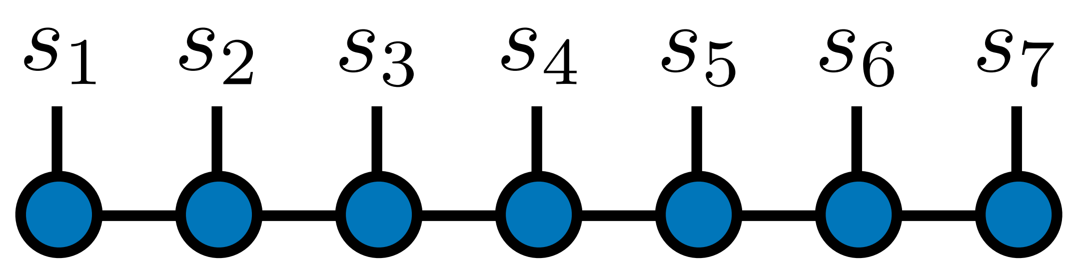
Tensor train (QTT) – one dimensional continuous function

$$f(x) =$$



Wavefunction – N-dimensional discrete function

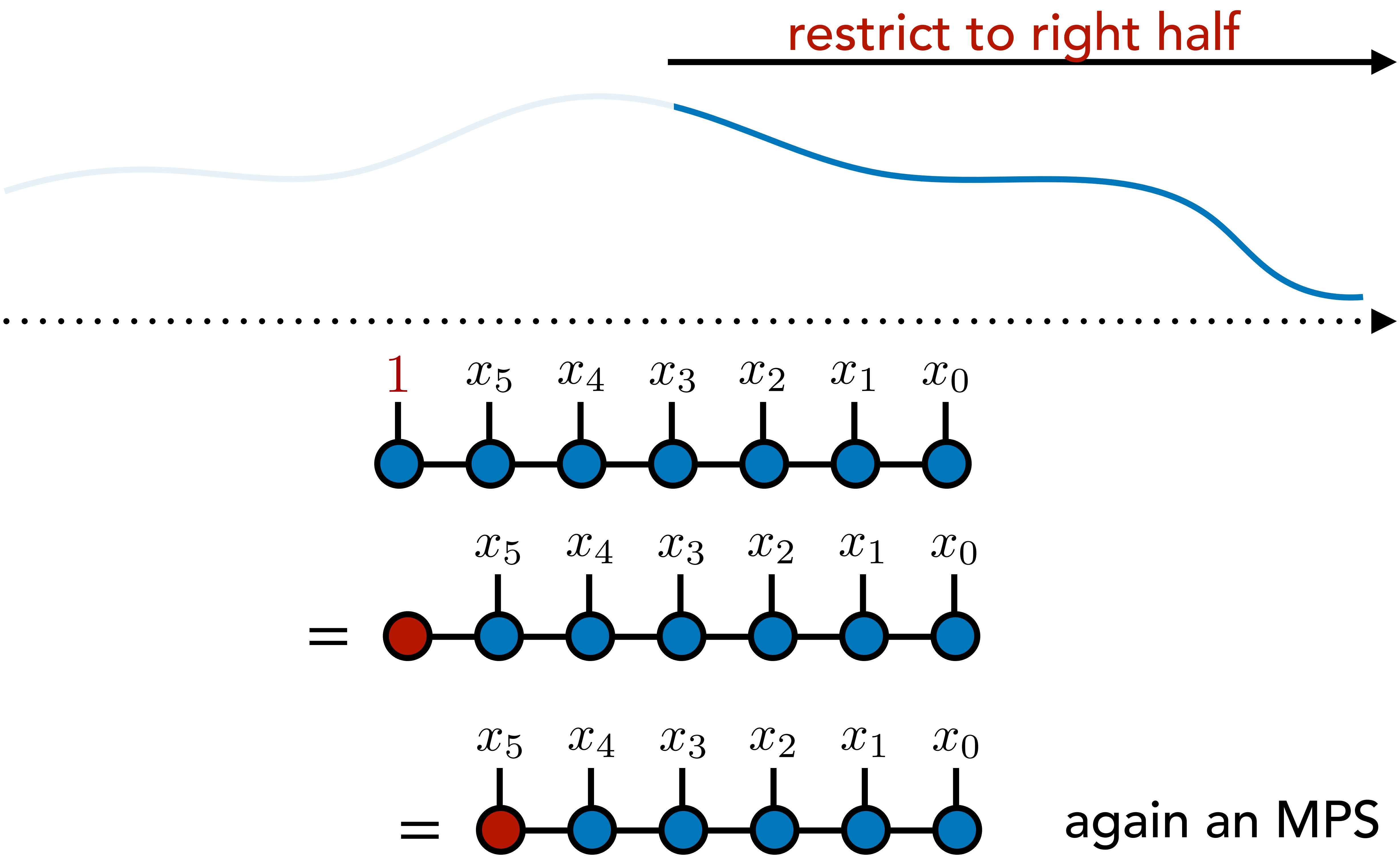
$$\Psi(s_1, s_2, s_3, s_4, s_5, s_6, s_7) =$$



Same tensor network, different interpretation

When does it work?

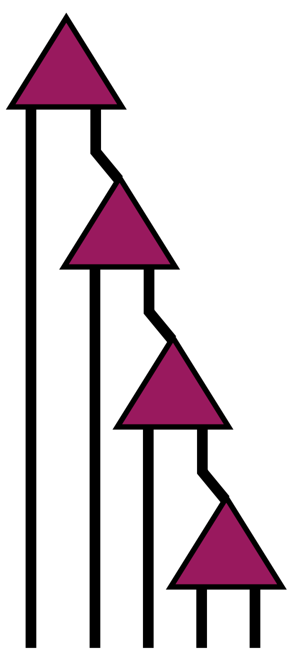
MPS function again an MPS when restricted:



When does it work?

MPS function again an MPS when restricted:

- implies self-similarity property
- includes smooth functions as special case
- can handle some amount of cusps & discontinuities too
- likely connection to wavelets, but differences too (e.g. adaptivity)*



*I.V. Oseledets, E.E. Tyrtyshnikov, SIAM Journal on Scientific Computing, 33(3), 1315

Crucially, nearly entire quantum tensor network toolbox (ITensor software) can be repurposed:

- eigenvector finding (DMRG and DMRG-X algorithms)
- time-dependent diff. eq. solving (tDMRG & TDVP algs.)
- solving linear systems
- discrete Fourier transform in compressed space
- and more...

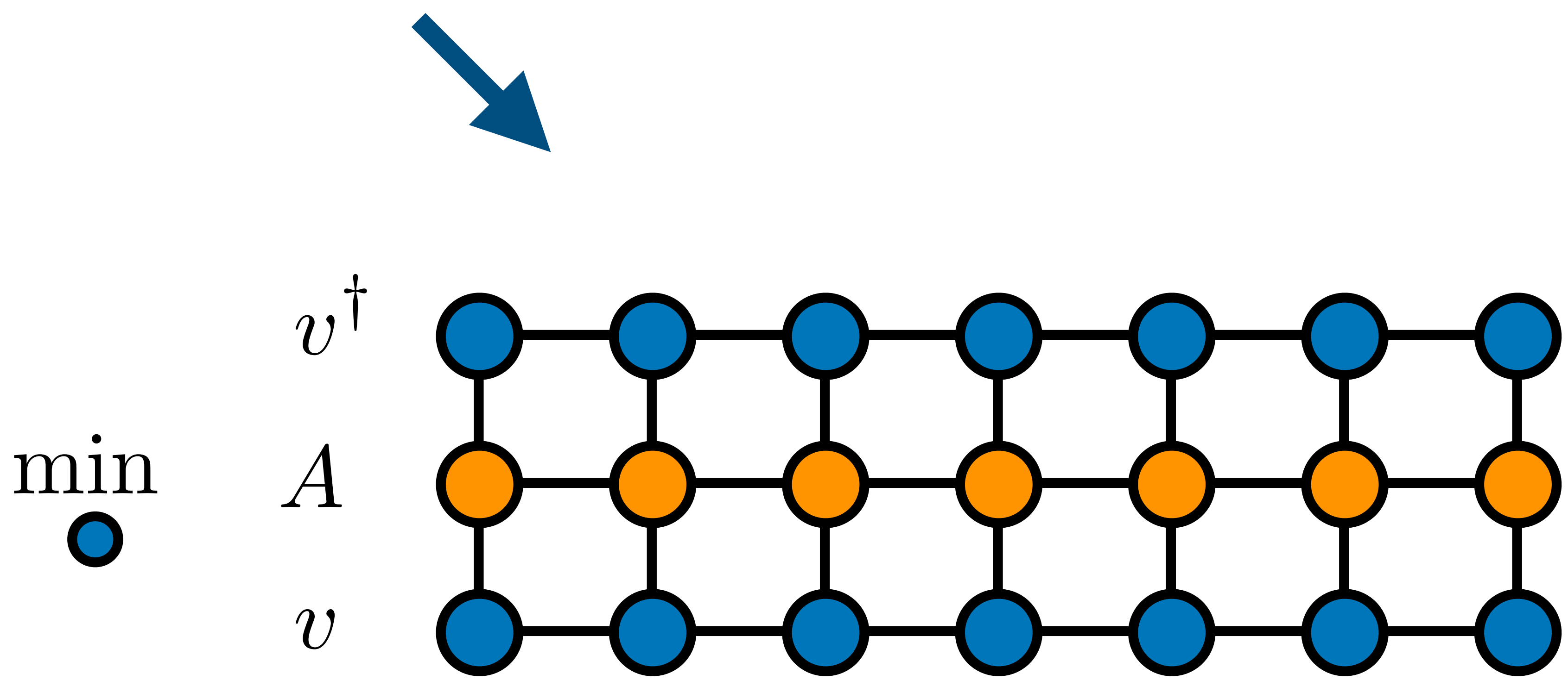
"Quantum inspired classical algorithms"

A quantum computer running on classical hardware

Example: Extremal Eigenvectors ("DMRG" algorithm)

Solve for extremal eigenvector $Av = \lambda v$
by minimizing Rayleigh quotient

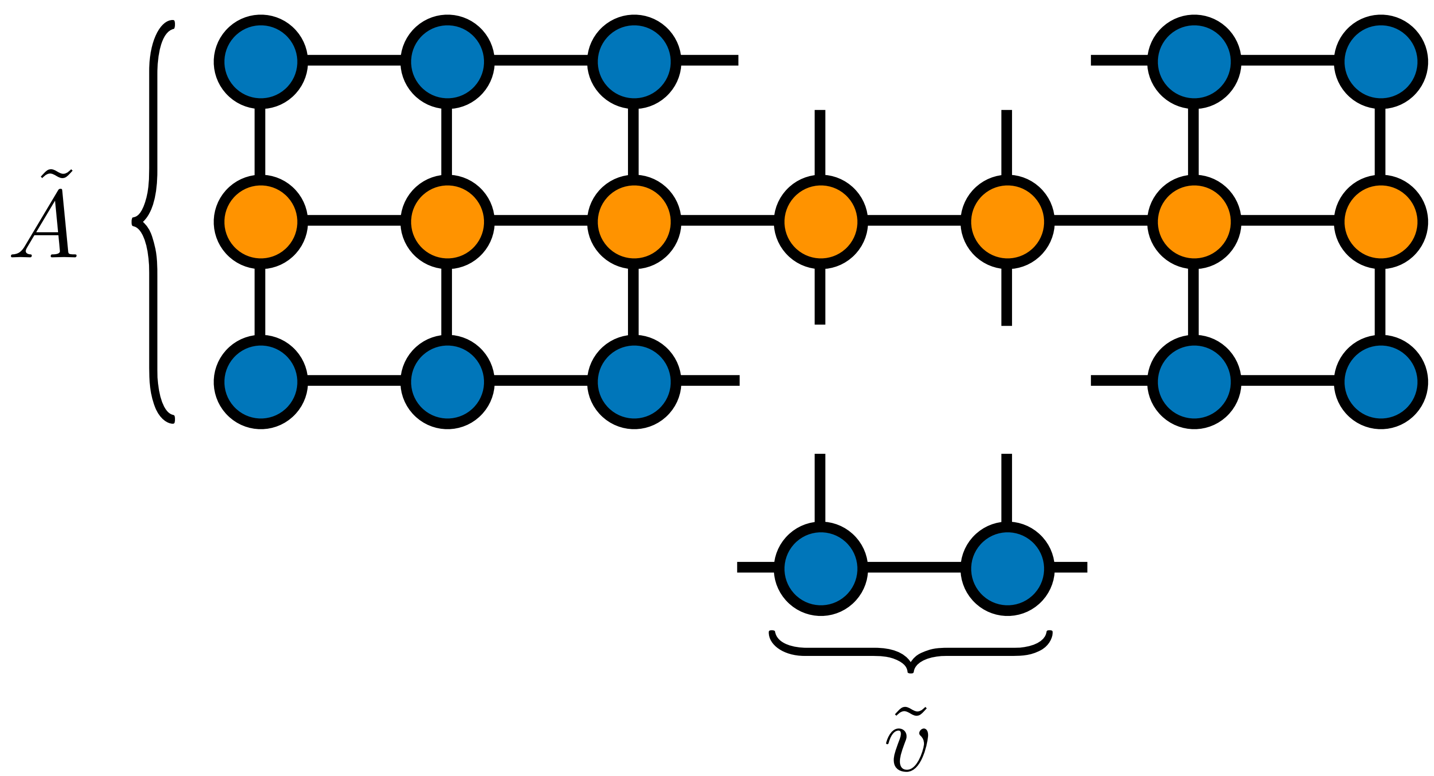
$$\min_{v} \quad \frac{v^\dagger A v}{v^\dagger v}$$



Example: Extremal Eigenvectors ("DMRG" algorithm)

Alternating strategy:

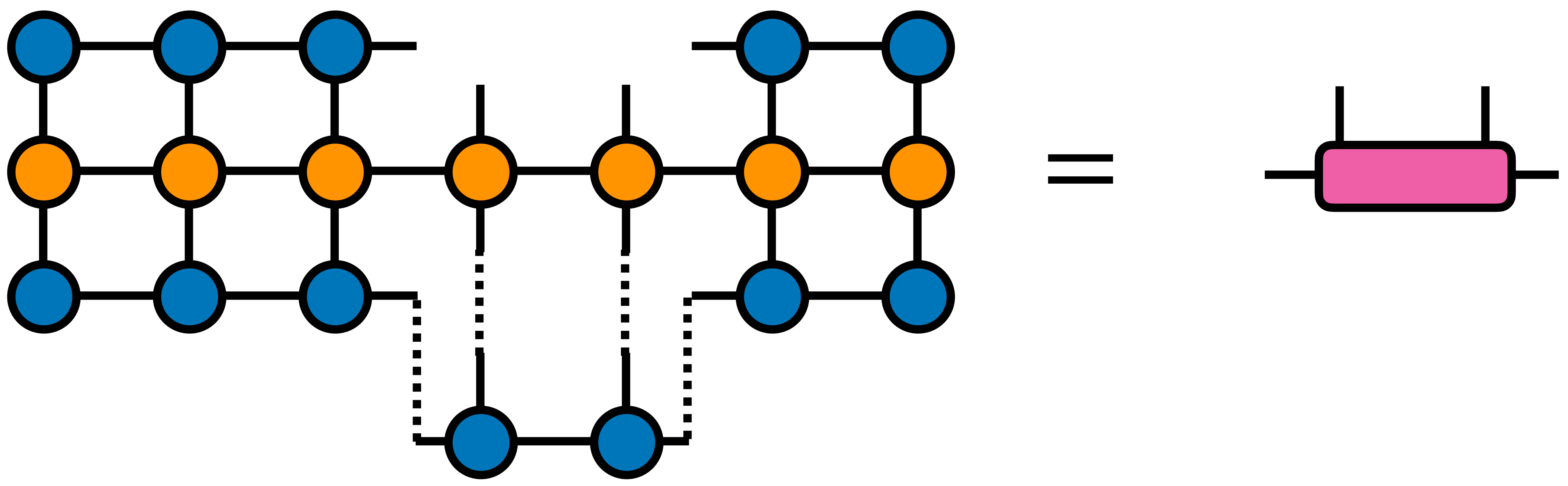
- freeze all but two tensors
- use remaining network as linear map in Krylov eigensolver



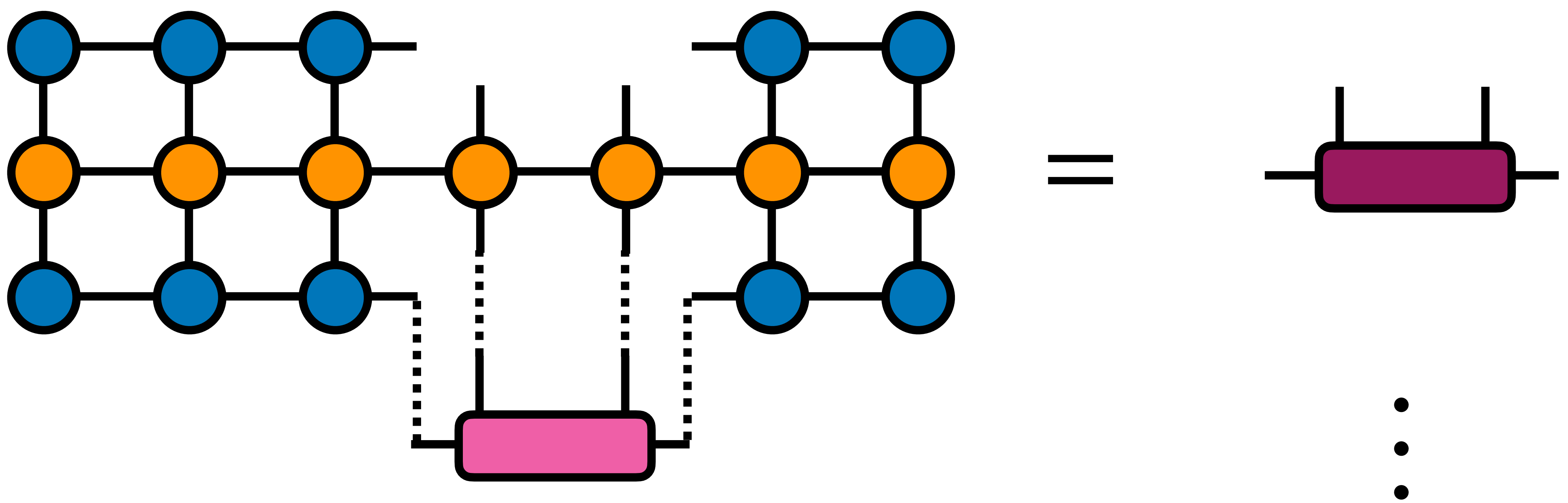
Example: Extremal Eigenvectors ("DMRG" algorithm)

Eigensolver iterations

(1)

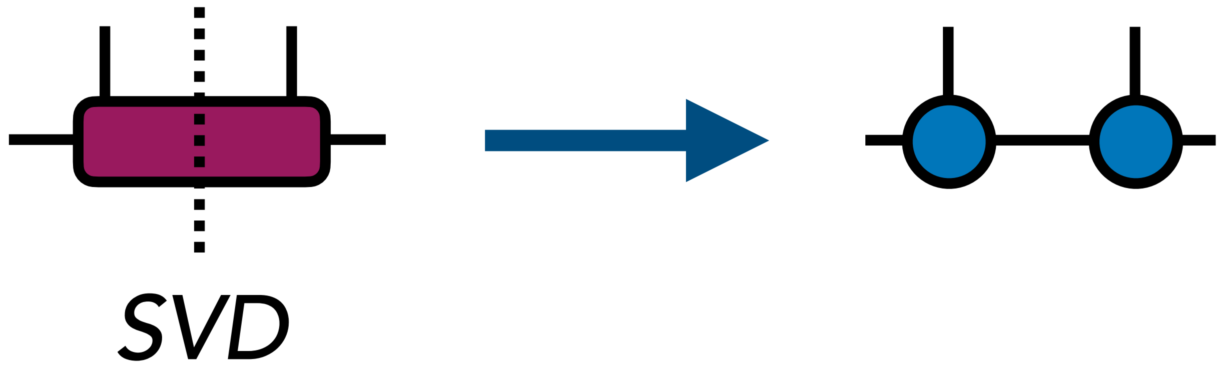


(2)



Example: Extremal Eigenvectors ("DMRG" algorithm)

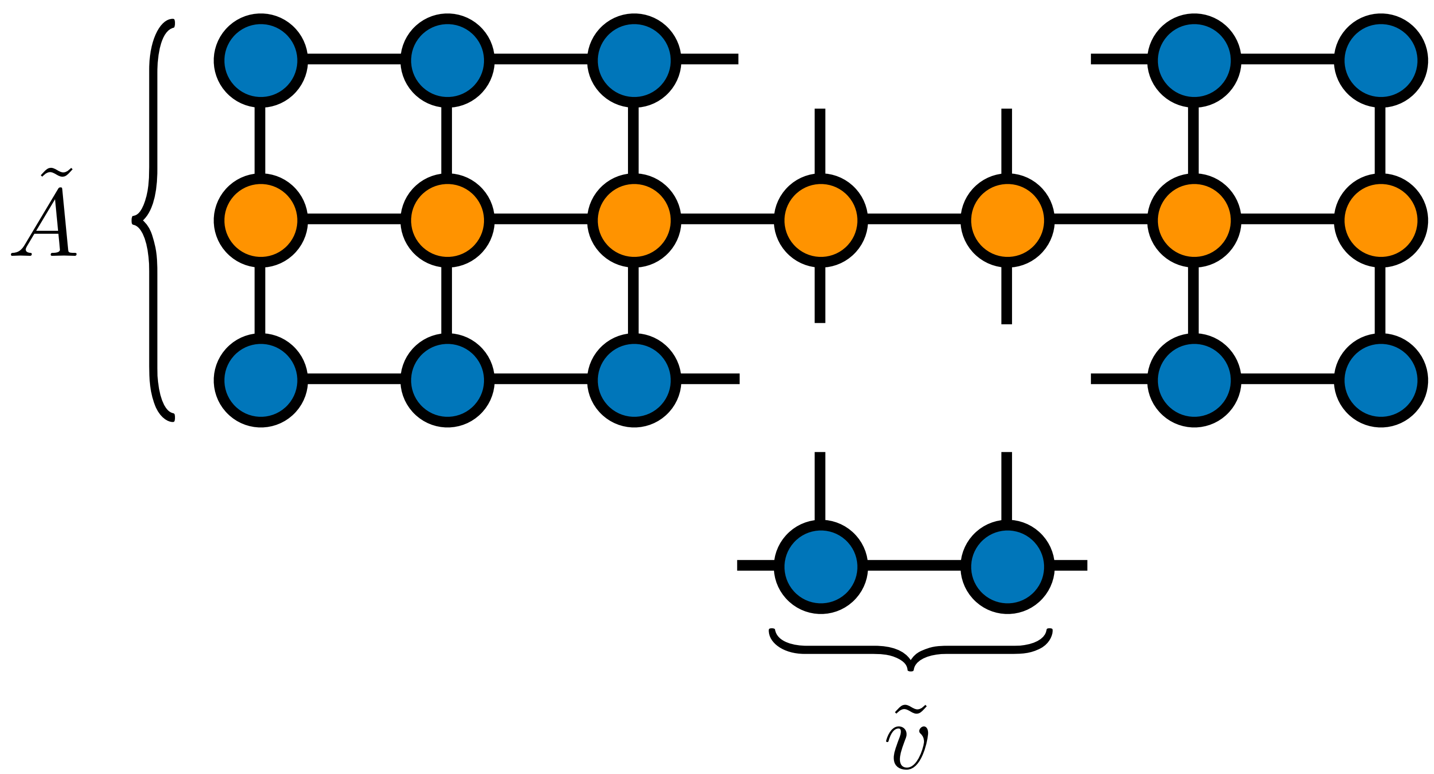
When done improving eigenvector,
use singular value decomposition (SVD) to restore MPS form
and adapt rank



Example: Extremal Eigenvectors ("DMRG" algorithm)

Benefits of method:

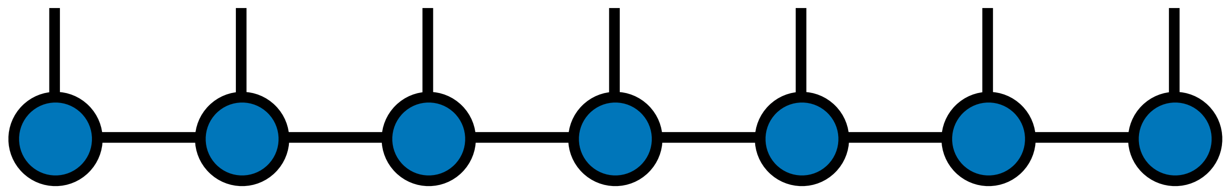
- adaptively determines internal ranks of MPS
- efficient: scaling $nd^2\chi^3$
- as few as 4-5 outer iteration ("sweeps") often enough



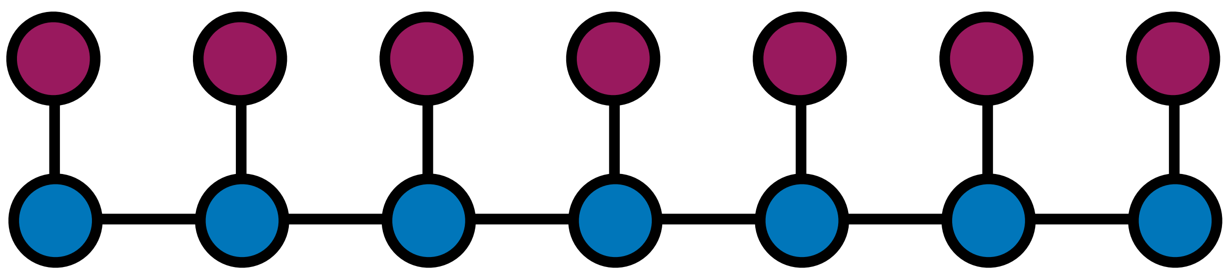
Applications

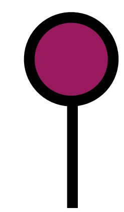
Function Integration

Given a function in compressed form

$f(x) \approx$ 

Straightforwardly compute its integral as

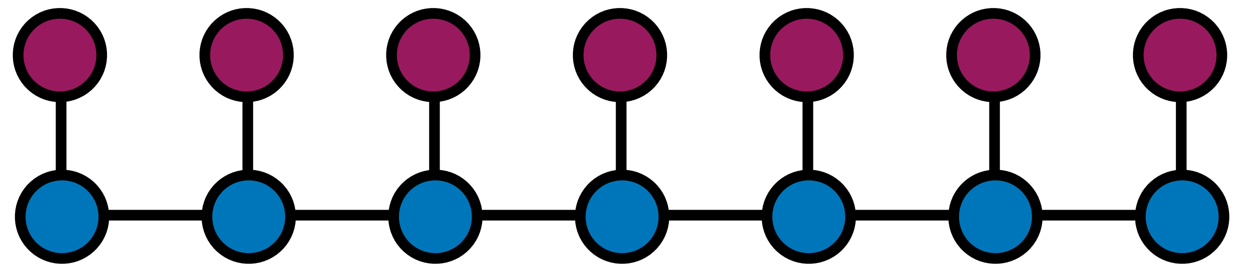
$\int_0^1 dx \; f(x) \approx$ 

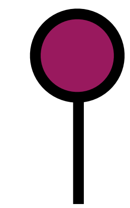
 $= \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}$

$$= \frac{1}{2^n} \sum_{x_0, x_1, \ldots, x_n} f(x_0, x_1, \ldots, x_n)$$

Function Integration

$$\int_0^1 dx\ f(x) \approx$$



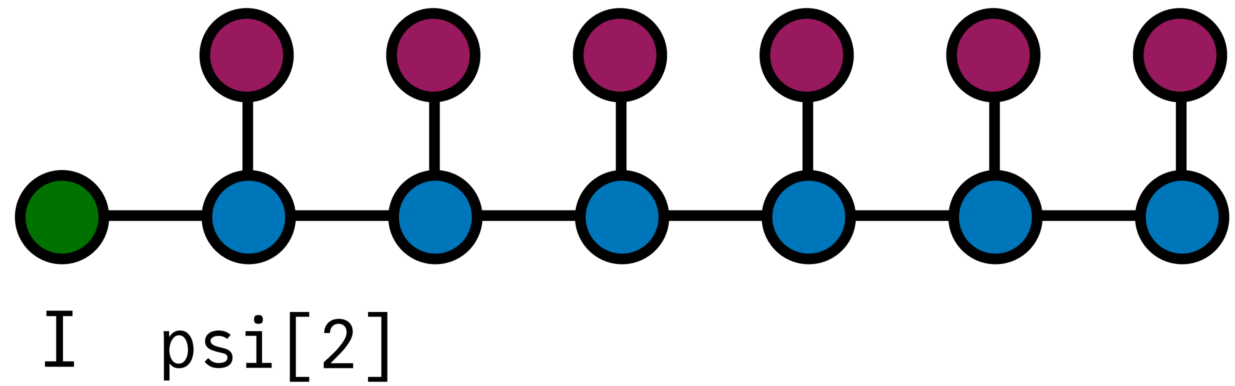


$$= \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}$$

ITensor code to perform integral:

```
function integrate(psi::MPS)
  sites = siteinds(psi)
  I = ITensor(1.)
  for (j,s) in enumerate(sites)
    I *= (psi[j]*ITensor([1/2,1/2],s))
  end
  return scalar(I)
end
```

Function Integration

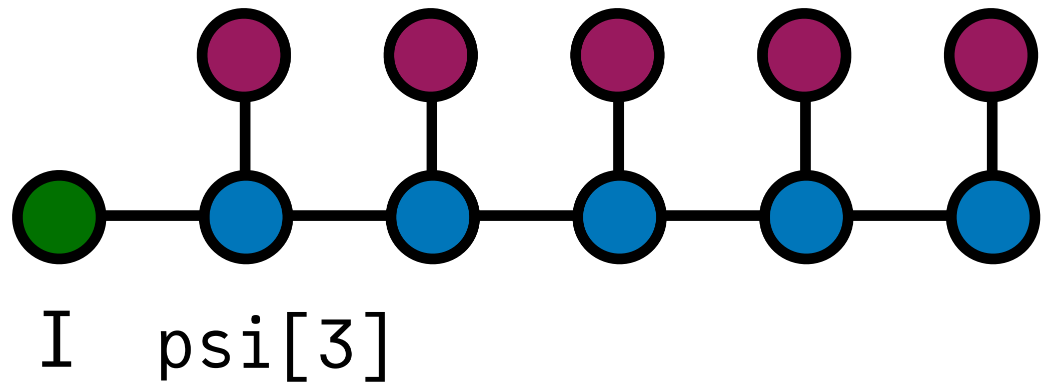


$$\approx \int_0^1 dx\ f(x)$$

ITensor code to perform integral:

```
function integrate(psi::MPS)
  sites = siteinds(psi)
  I = ITensor(1.)
  for (j,s) in enumerate(sites)
    I *= (psi[j]*ITensor([1/2,1/2],s))
  end
  return scalar(I)
end
```

Function Integration

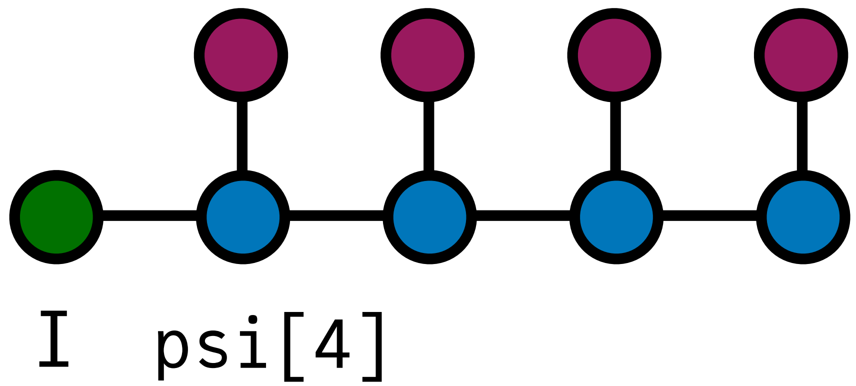


$$\approx \int_0^1 dx\ f(x)$$

ITensor code to perform integral:

```
function integrate(psi::MPS)
  sites = siteinds(psi)
  I = ITensor(1.)
  for (j,s) in enumerate(sites)
    I *= (psi[j]*ITensor([1/2,1/2],s))
  end
  return scalar(I)
end
```

Function Integration

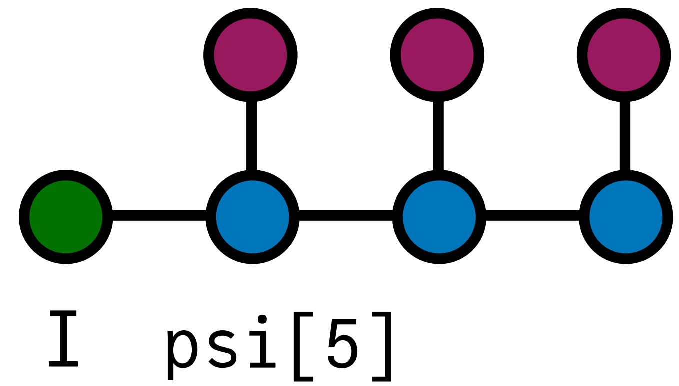


$$\approx \int_0^1 dx\ f(x)$$

ITensor code to perform integral:

```
function integrate(psi::MPS)
  sites = siteinds(psi)
  I = ITensor(1.)
  for (j,s) in enumerate(sites)
    I *= (psi[j]*ITensor([1/2,1/2],s))
  end
  return scalar(I)
end
```

Function Integration

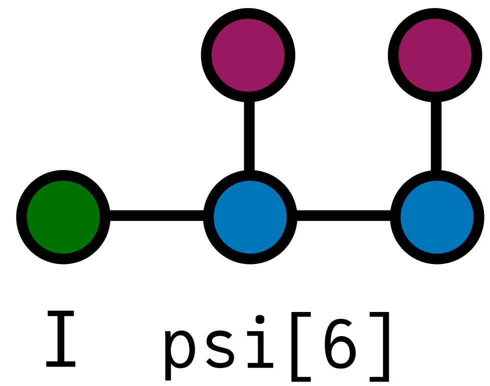


$$\approx \int_0^1 dx\ f(x)$$

ITensor code to perform integral:

```
function integrate(psi::MPS)
  sites = siteinds(psi)
  I = ITensor(1.)
  for (j,s) in enumerate(sites)
    I *= (psi[j]*ITensor([1/2,1/2],s))
  end
  return scalar(I)
end
```

Function Integration

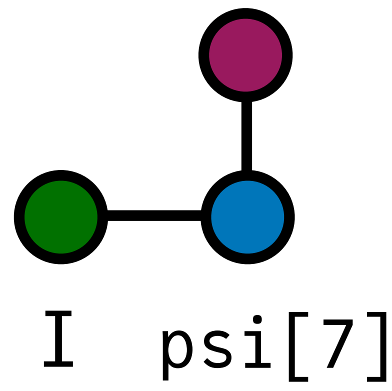


$$\approx \int_0^1 dx\ f(x)$$

ITensor code to perform integral:

```
function integrate(psi::MPS)
  sites = siteinds(psi)
  I = ITensor(1.)
  for (j,s) in enumerate(sites)
    I *= (psi[j]*ITensor([1/2,1/2],s))
  end
  return scalar(I)
end
```

Function Integration

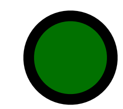


$$\approx \int_0^1 dx\ f(x)$$

ITensor code to perform integral:

```
function integrate(psi::MPS)
  sites = siteinds(psi)
  I = ITensor(1.)
  for (j,s) in enumerate(sites)
    I *= (psi[j]*ITensor([1/2,1/2],s))
  end
  return scalar(I)
end
```

Function Integration

 $\approx \int_0^1 dx\ f(x)$

ITensor code to perform integral:

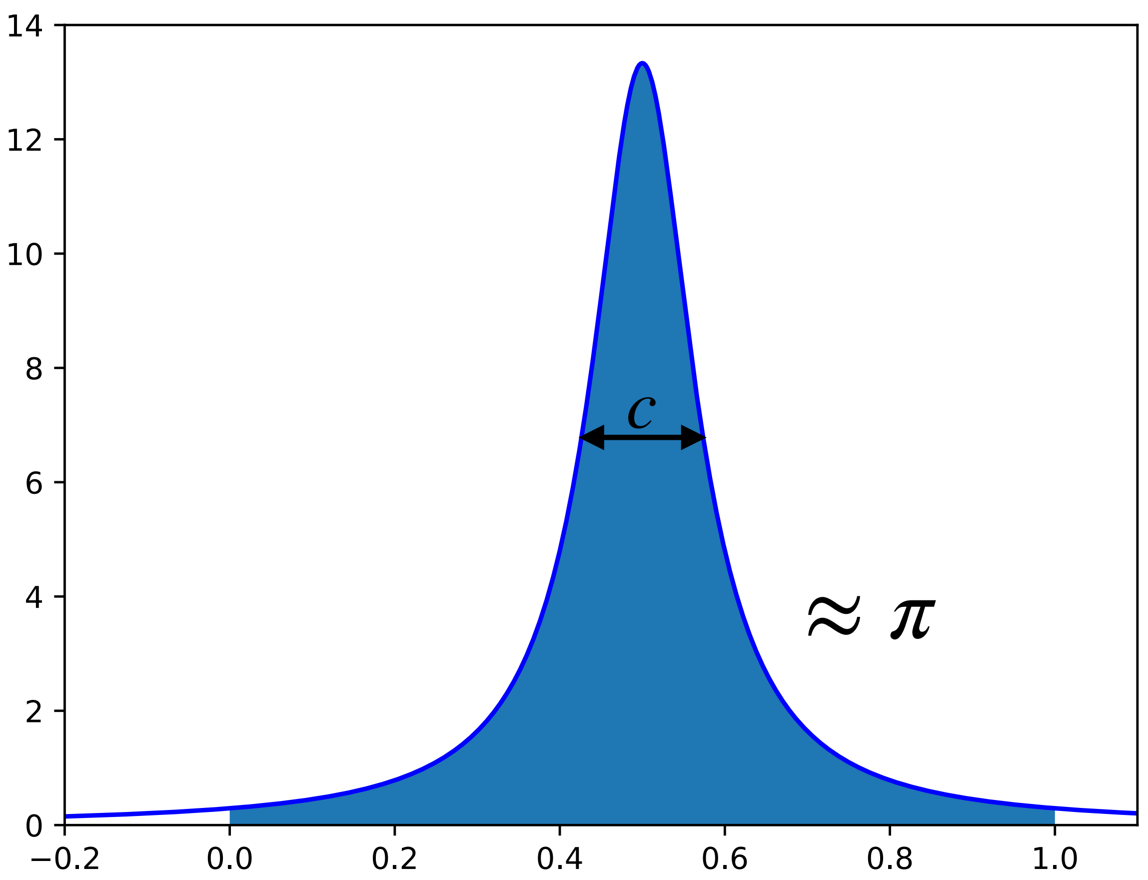
```
function integrate(psi::MPS)
  sites = siteinds(psi)
  I = ITensor(1.)
  for (j,s) in enumerate(sites)
    I *= (psi[j]*ITensor([1/2,1/2],s))
  end
  return scalar(I)
end
```

Function Integration

Test case – unnormalized Cauchy distribution

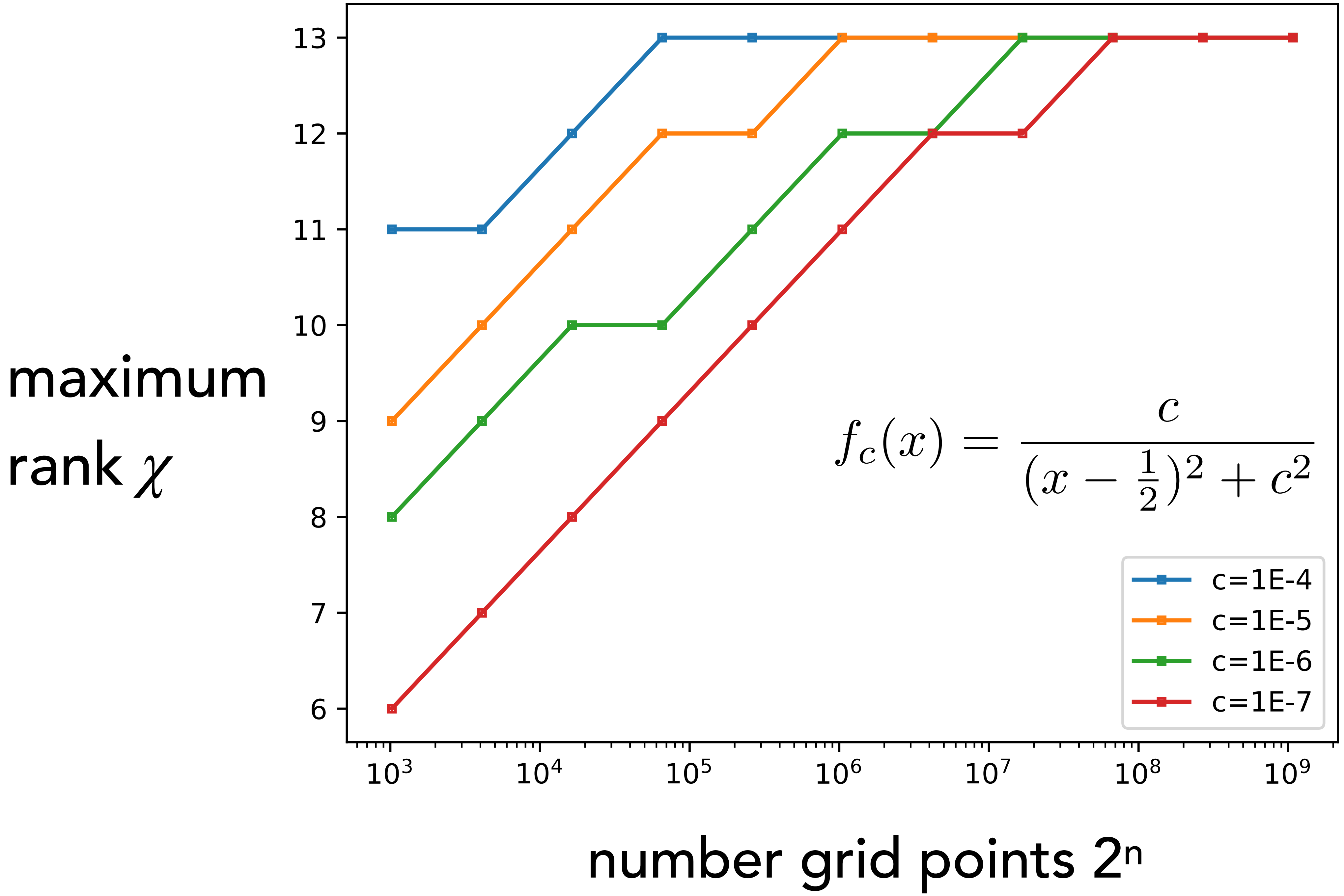
$$f_c(x) = \frac{c}{(x-\frac{1}{2})^2 + c^2}$$

such that $$\lim_{c\to 0} \int_0^1 dx \; f(x) = \pi$$



Function Integration

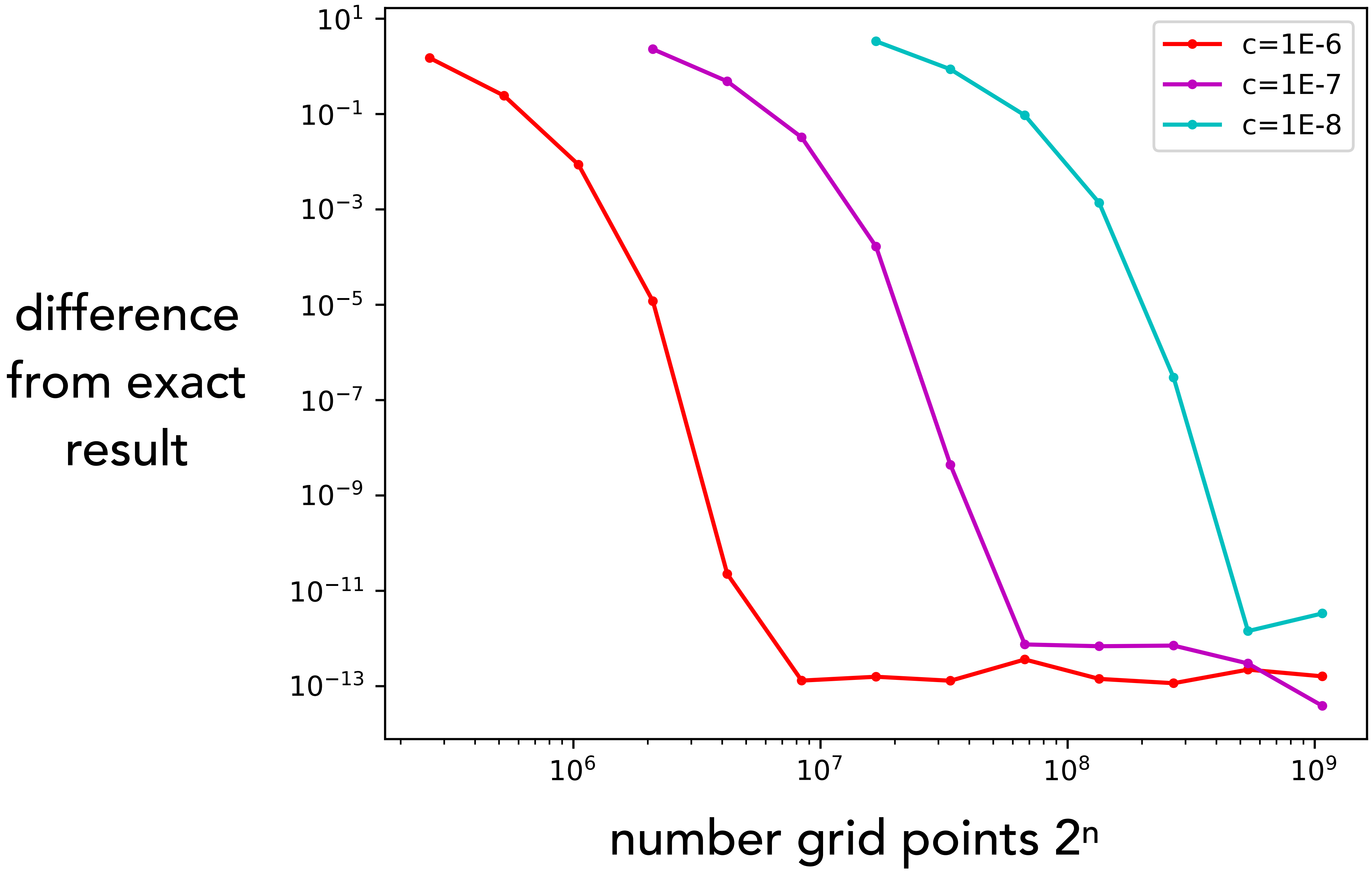
Maximum ranks as function of grid size:



Function Integration

Results for fixed c values

$$\int_0^1 \frac{c}{(x-\frac{1}{2})^2 + c^2} dx$$

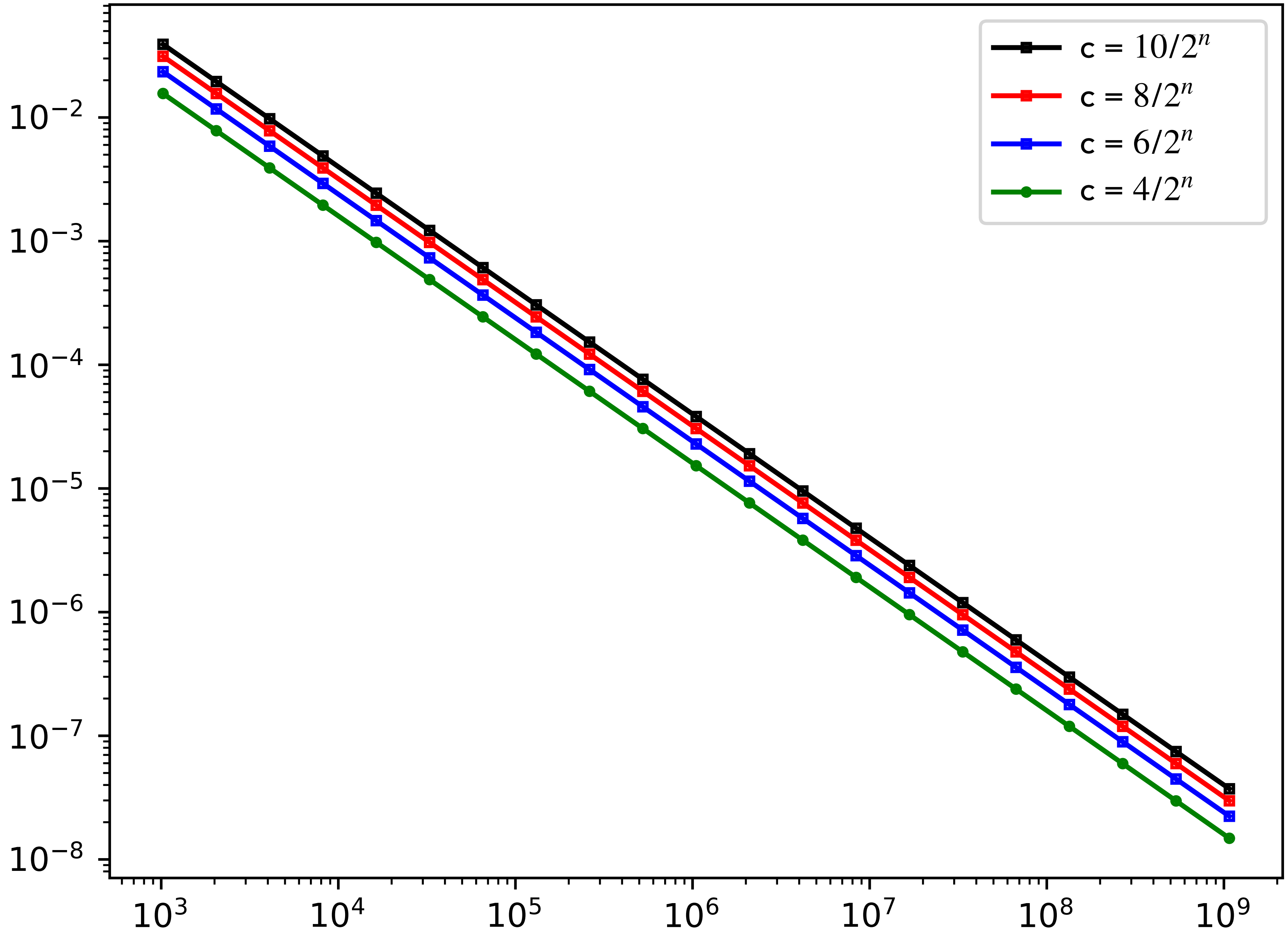


Function Integration

$$\int_0^1 \frac{c}{(x-\frac{1}{2})^2 + c^2} dx$$

Scaling c to zero (c $\sim 1/2^n$)
as function of grid spacing

difference from π



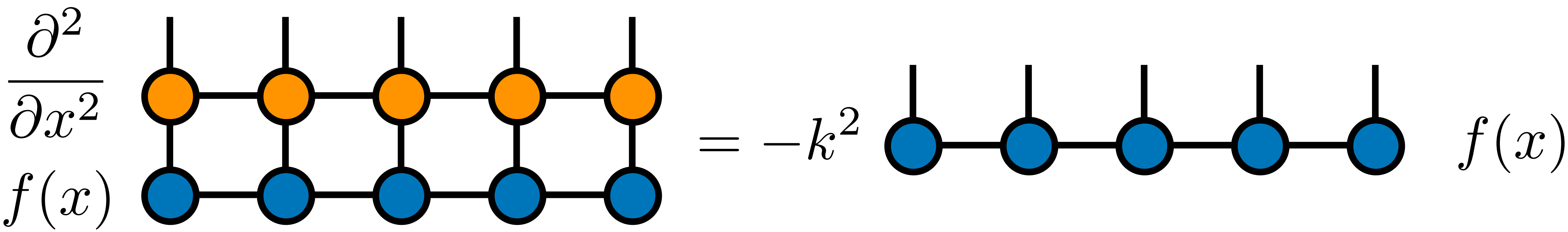
number grid points 2^n

Differential Equation Solving

Given a diff. eq. such as wave equation

$$\frac{\partial^2}{\partial x^2} f(x) = -k^2 f(x)$$

Can encode as tensor network equation

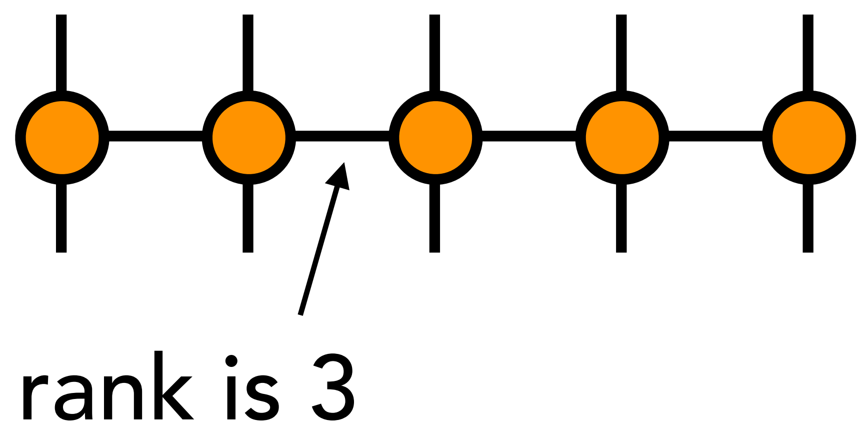


Use "DMRG-X" algorithm to efficiently find eigenvector

Differential Equation Solving

Finite-difference formula for $\dfrac{\partial^2}{\partial x^2}$

translates into exact expression for
low-rank *matrix product operator* (MPO)



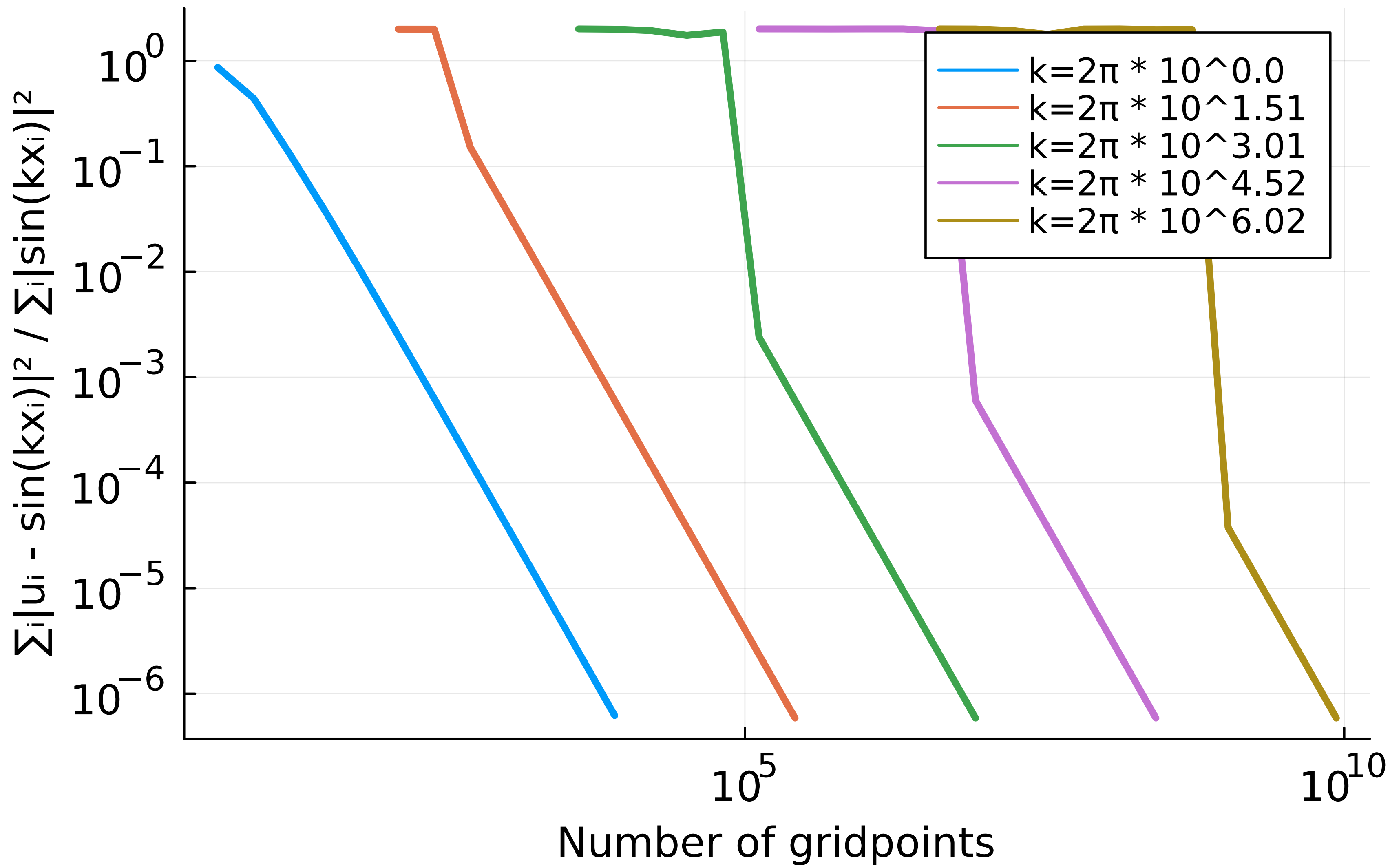
Basically: forward binary adder, backwards binary adder,
plus constant = $(x_{j+1} - 2x_j + x_{j-1})/a^2$

Differential Equation Solving

Solutions to wave equation

Use DMRG-X to solve (eigenvector with specific k)

Difference to exact solution



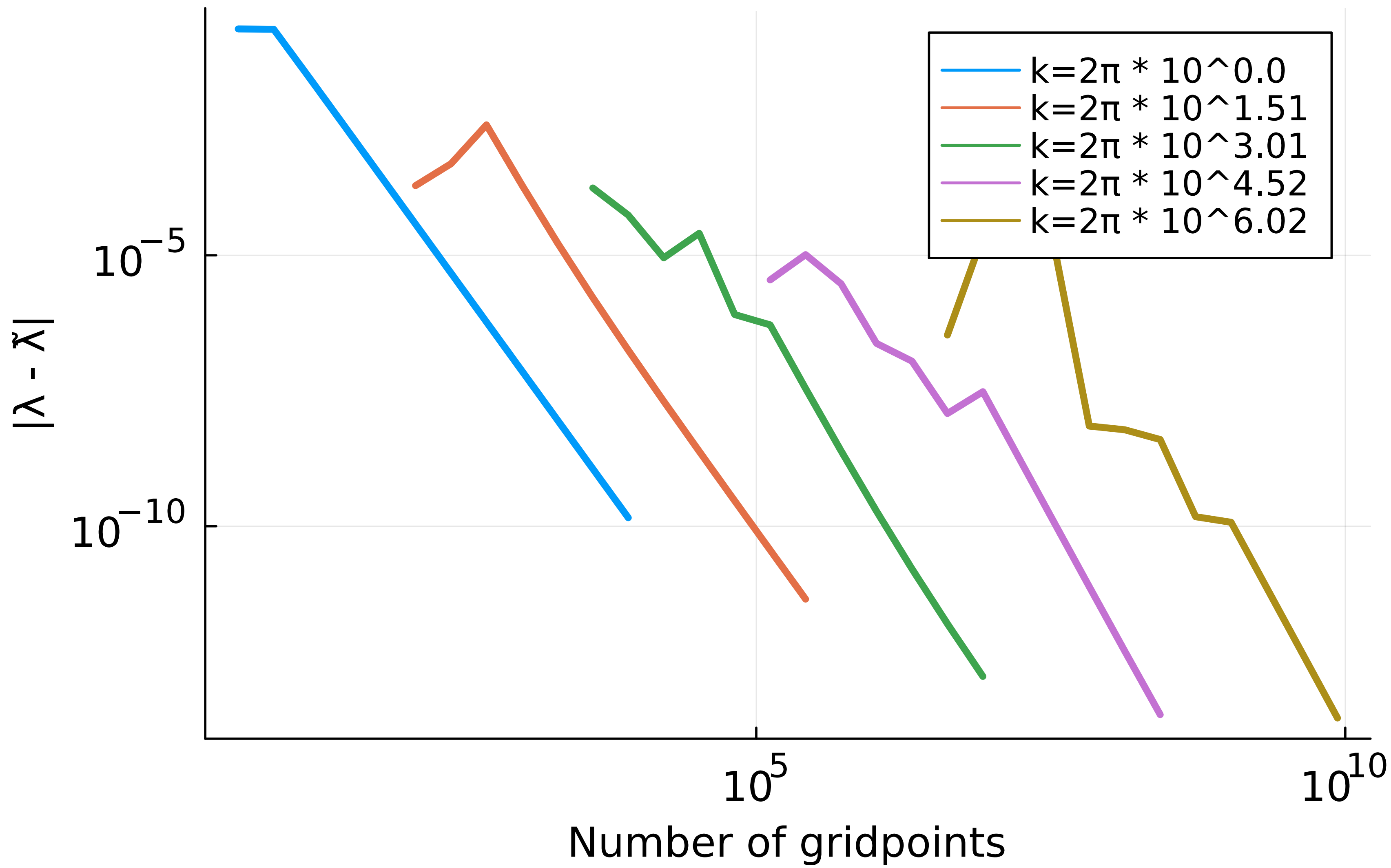
Reach frequency $k=10^6$ and 10^{10} grid points

Differential Equation Solving

Solutions to wave equation

Use DMRG-X to solve (eigenvector with specific k)

Eigenvalue error



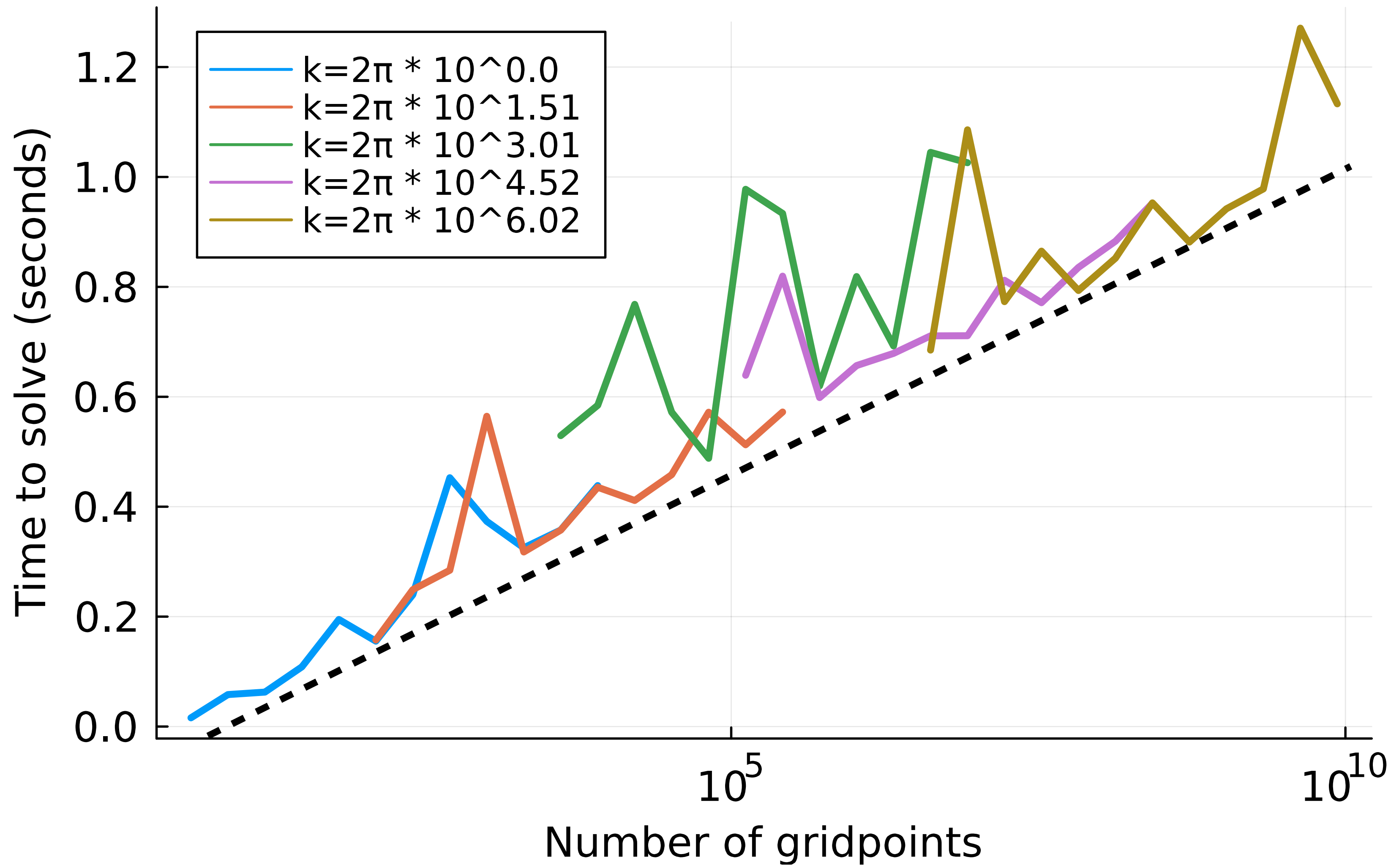
Reach frequency $k=10^6$ and 10^{10} grid points

Differential Equation Solving

Solutions to wave equation

Use DMRG-X to solve (eigenvector with specific k)

Time to solution



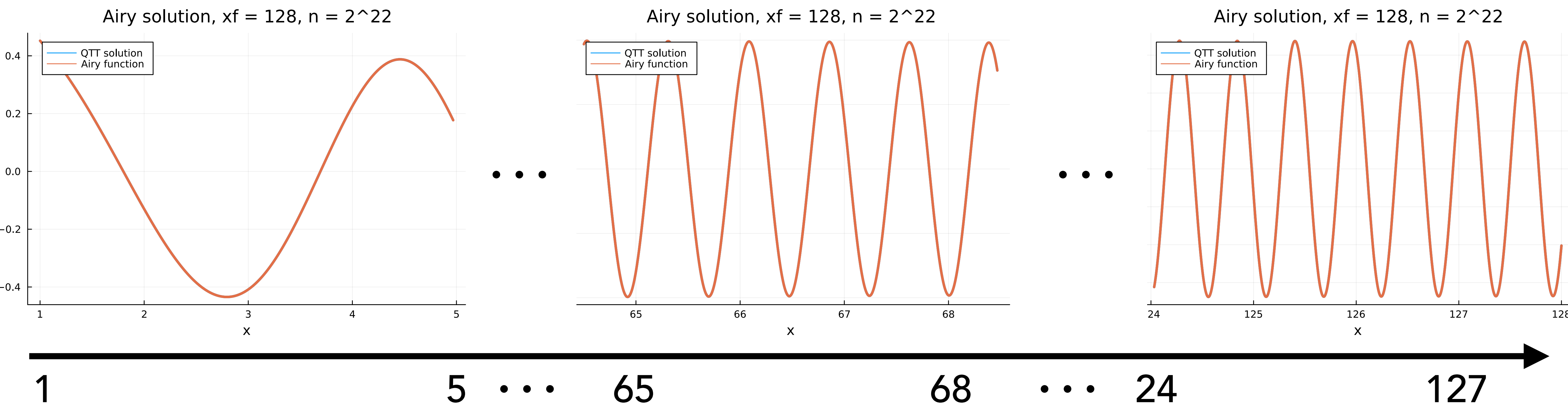
Logarithmic scaling with # grid points

MPS ranks all $\chi = 2$

Airy Differential Equation

Airy equation solutions are waves with frequency locally equal to position x

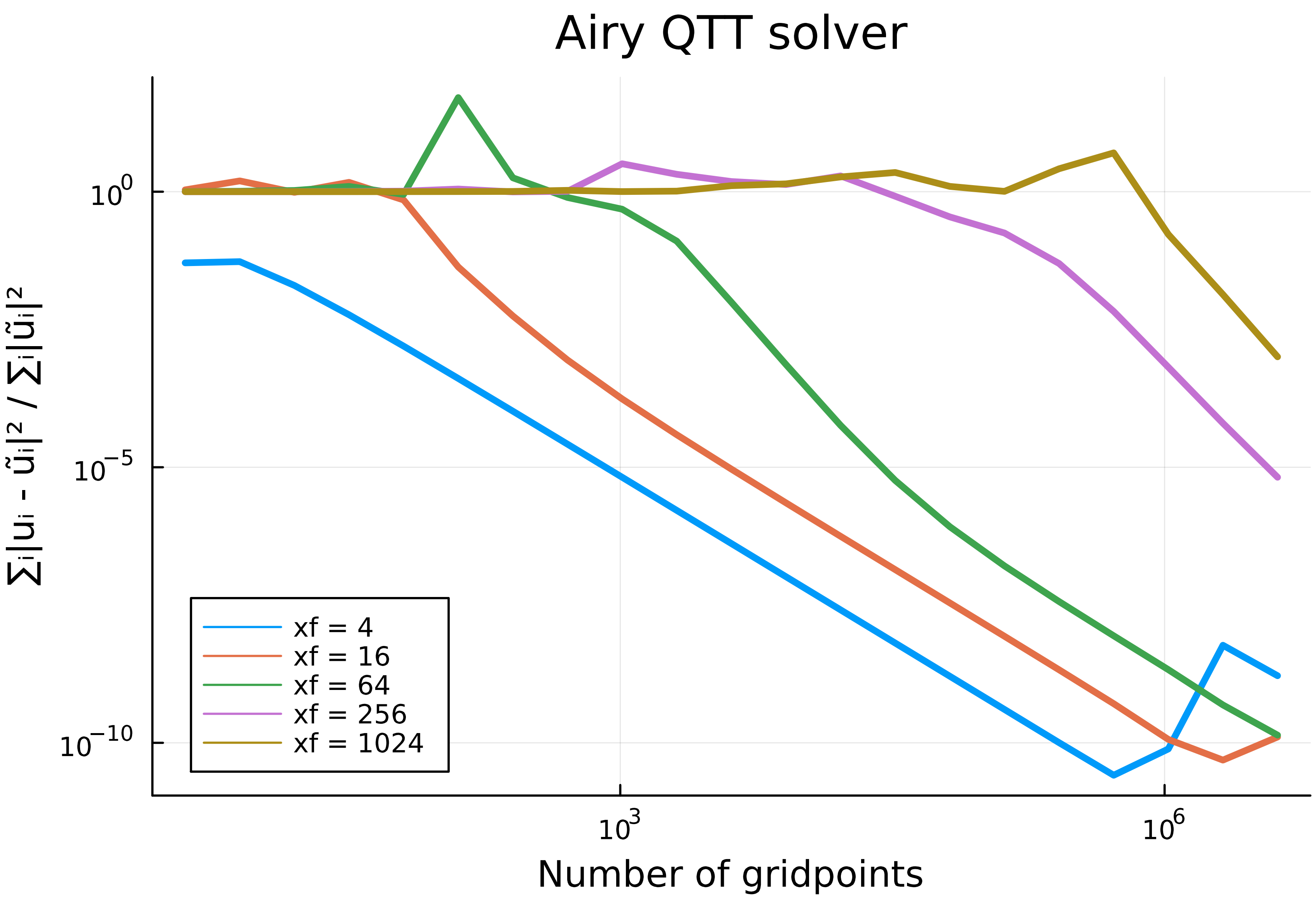
$$\frac{d^2 f(x)}{dx^2} = -x f(x)$$



Airy Differential Equation

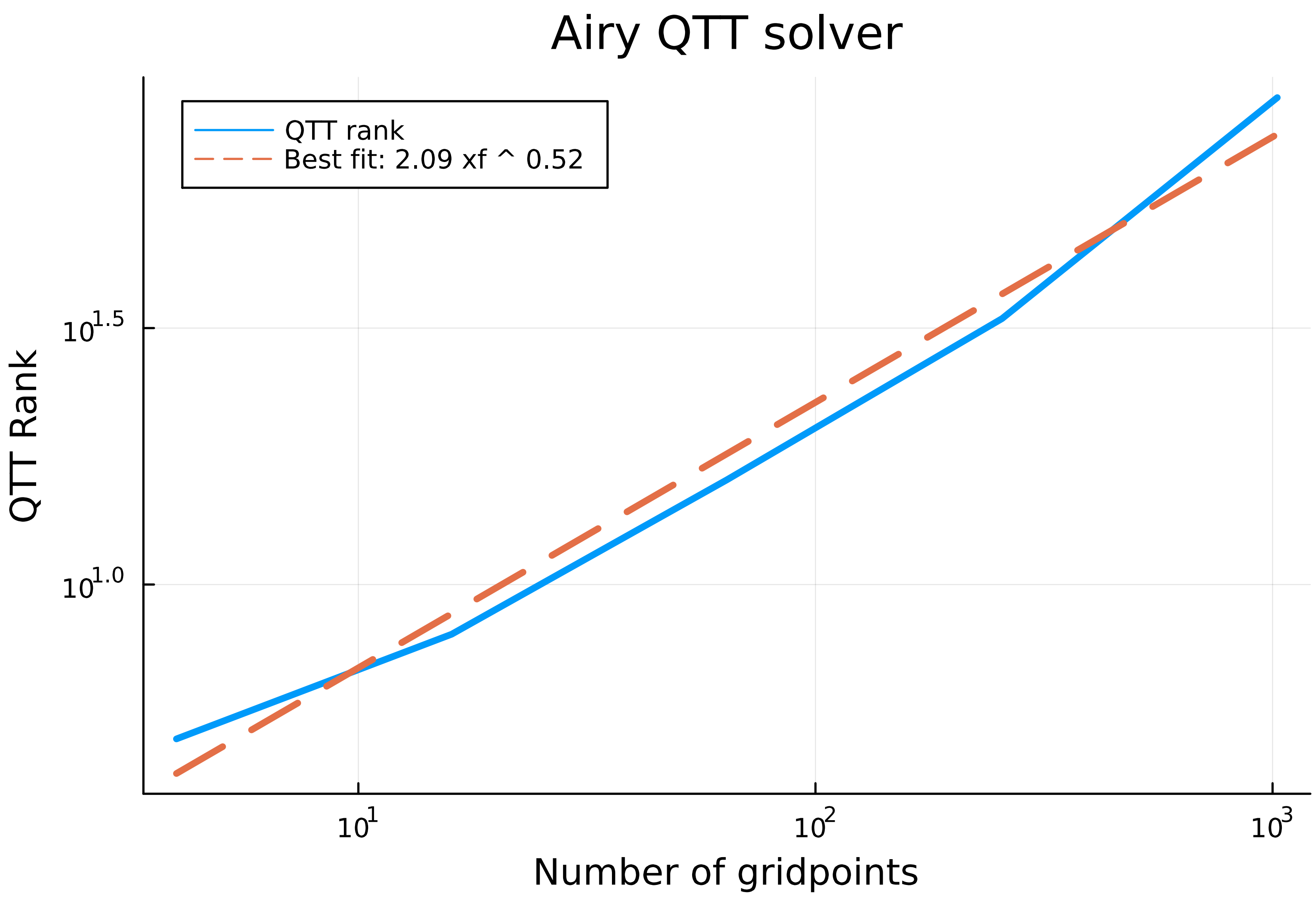
Solve as a boundary value problem from $x_i = 1$ to x_f using MPS linear solver

difference to exact solution



Airy Differential Equation

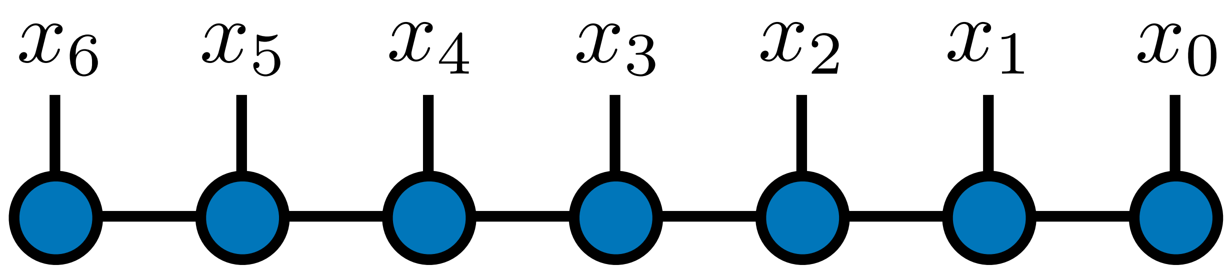
Scaling of ranks with problem size x_f is $\chi = (x_f)^{1/2}$



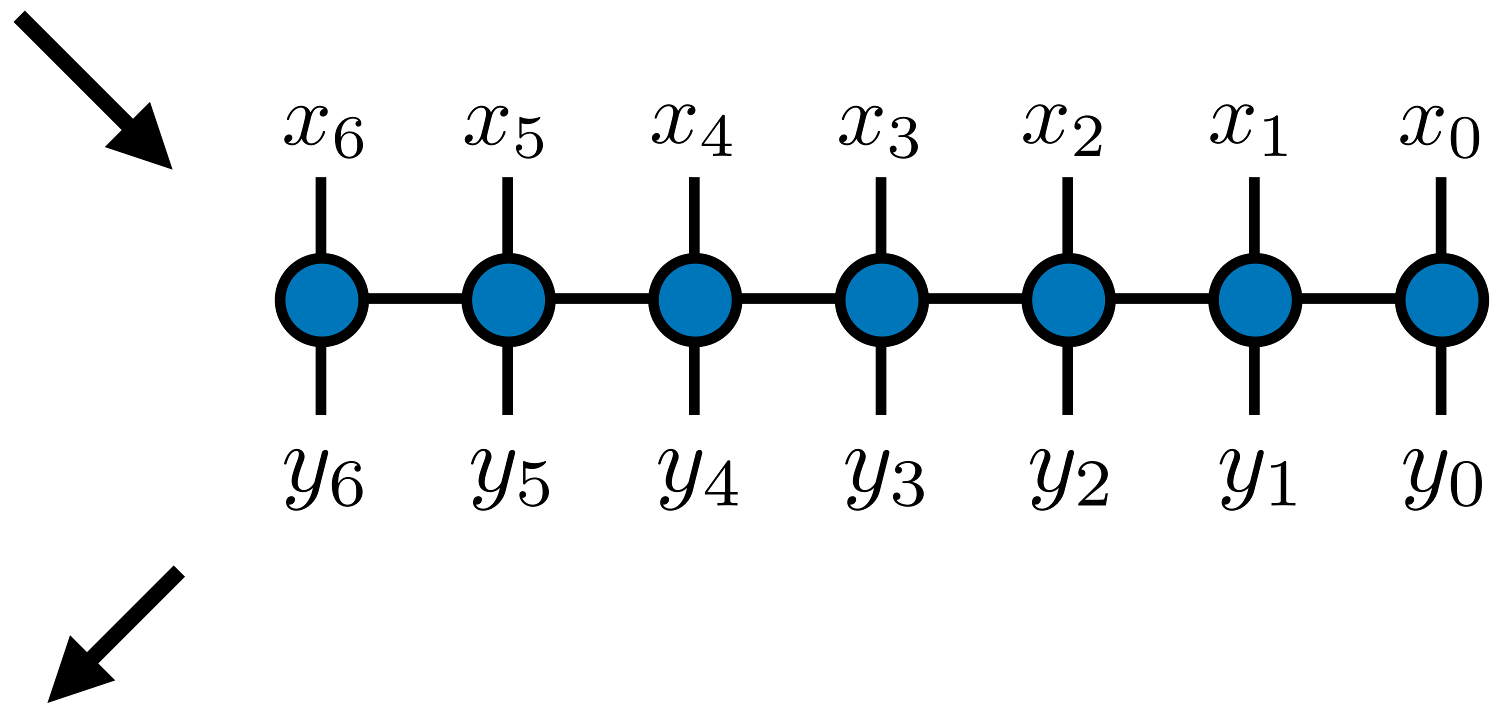
Memory scales as $\chi^2 = x_f$

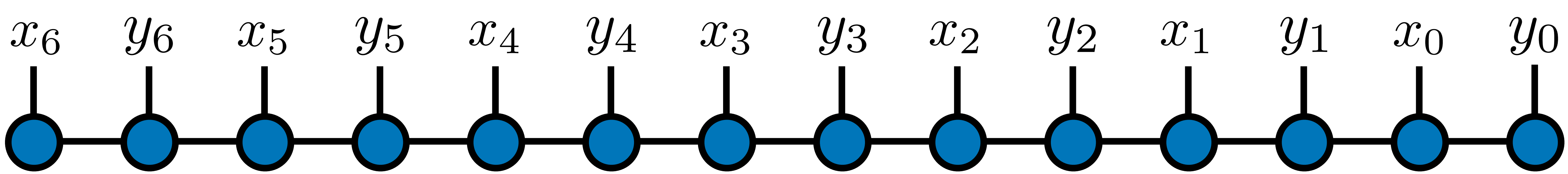
Effort scales at most $\chi^3 = (x_f)^{3/2}$

Higher-Dimensional Functions



To do two dimensions, just double tensor indices





MPS with 2N indices

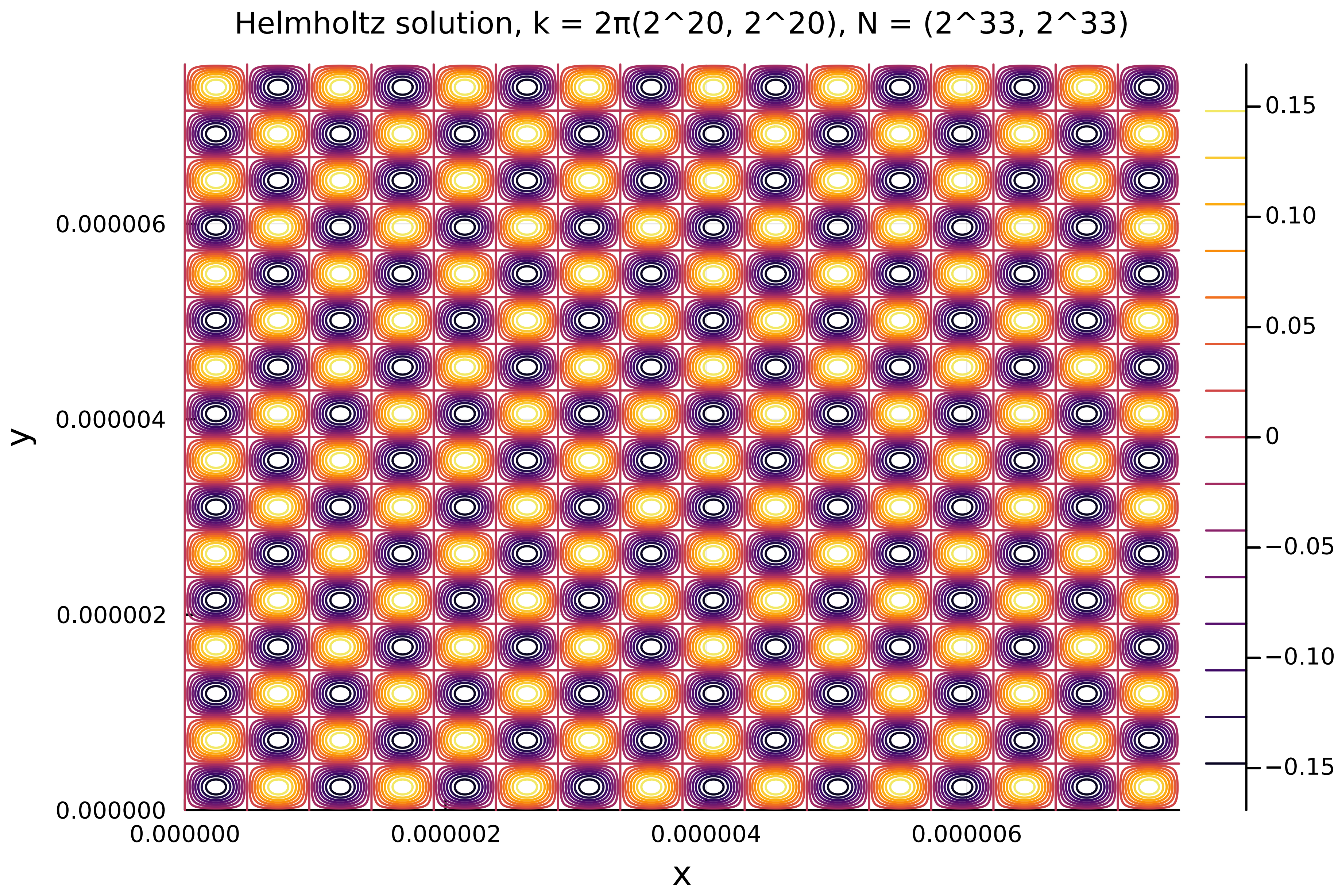
Higher-Dimensional Differential Equations

2D Helmholtz equation: $\nabla^2 f(x,y) = -k^2 f(x,y)$

Example result for 2D Helmholtz

Rank is uniform: $\chi = 4$

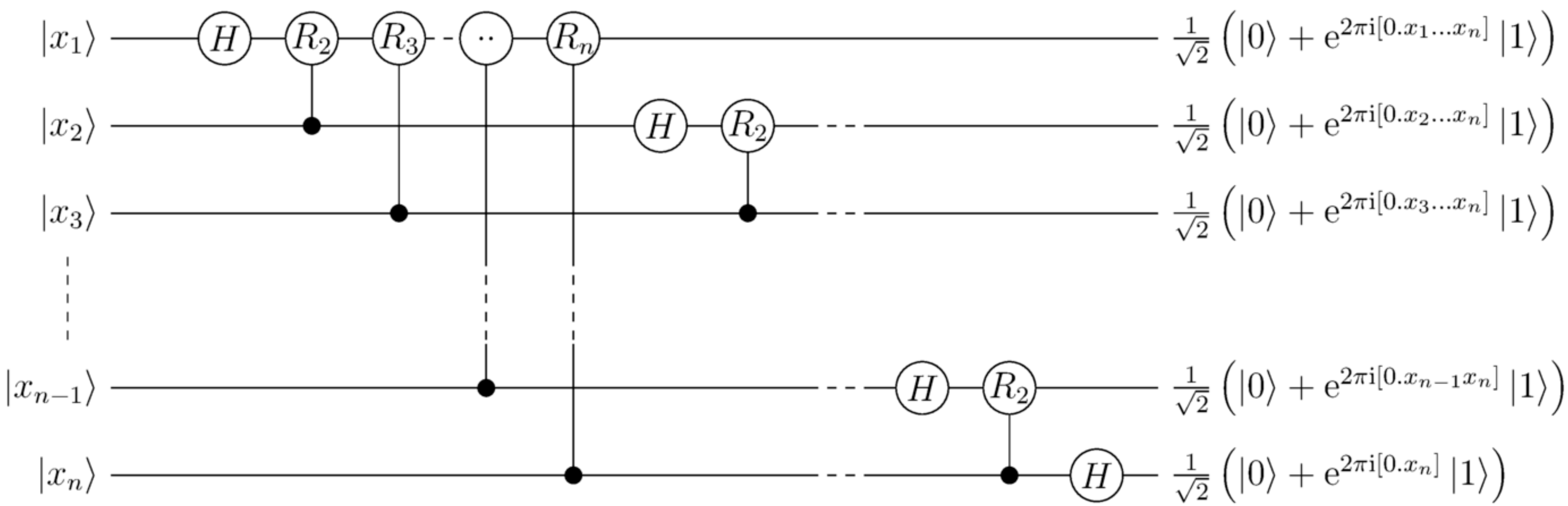
$f(x,y) \propto \sin(k_x x)\ \sin(k_y y)$



Function Analysis

Can analyze solutions without leaving compressed form

How? Quantum computing offers the quantum Fourier transform (QFT) circuit

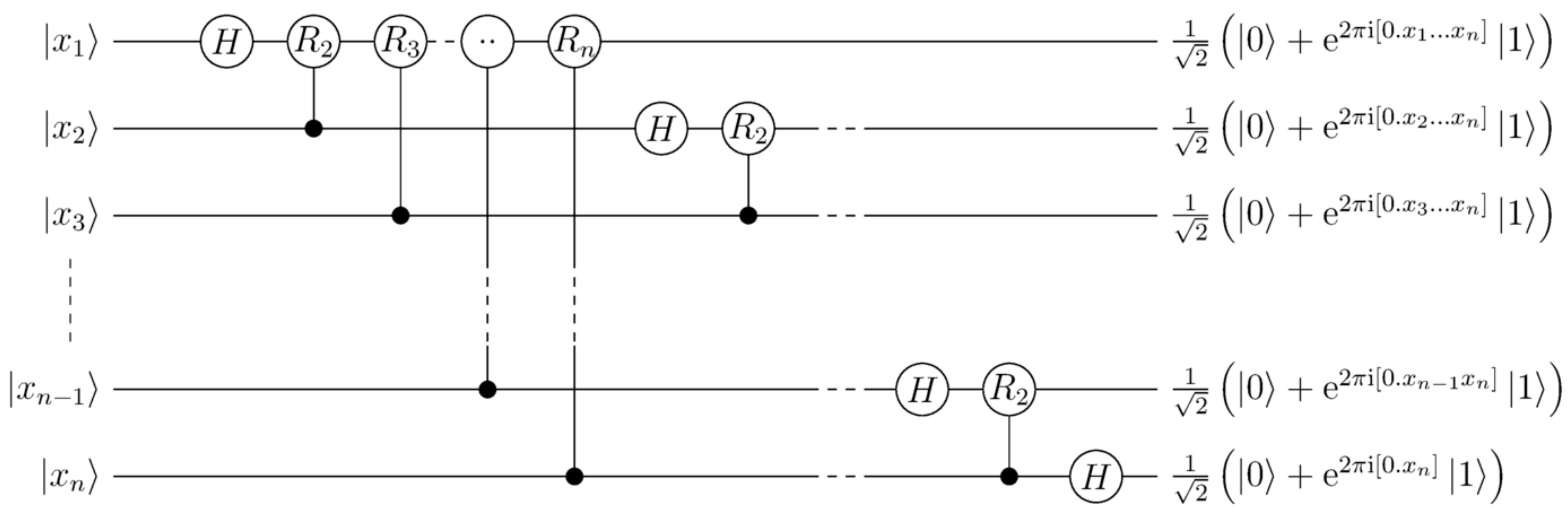


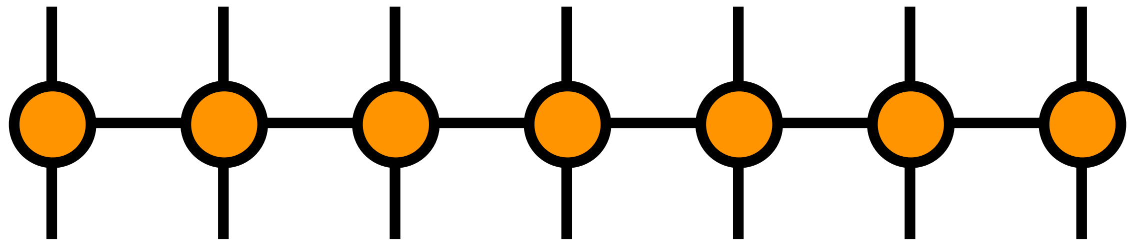
Coppersmith, D. (1994). "An approximate Fourier transform useful in quantum factoring". *Technical Report RC19642, IBM*. arXiv:quant-ph/0201067

Michael Nielsen and Isaac Chuang (2000). *Quantum Computation and Quantum Information*. Cambridge: Cambridge University Press

Function Analysis

QFT circuit meant for quantum computers



$\approx$ 

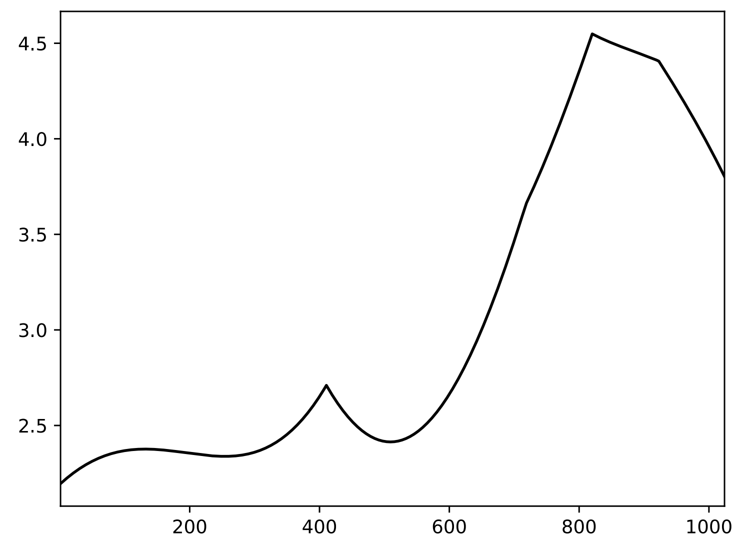
Turns out to be MPO tensor network of rank $\chi = \underline{\mathbf{8}}$!

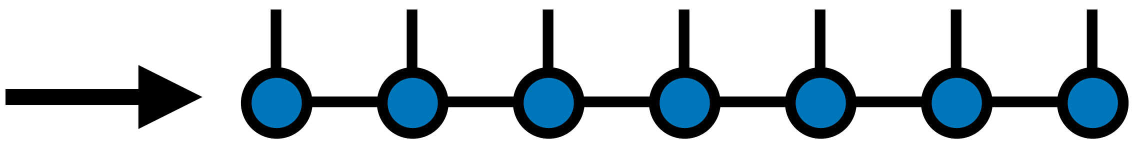
(Independent of grid size)

Function Analysis

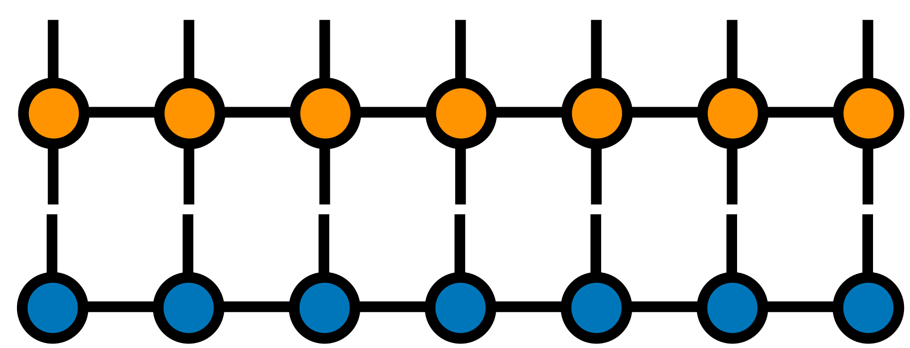
Can use to perform "superfast" Fourier transform

compress function



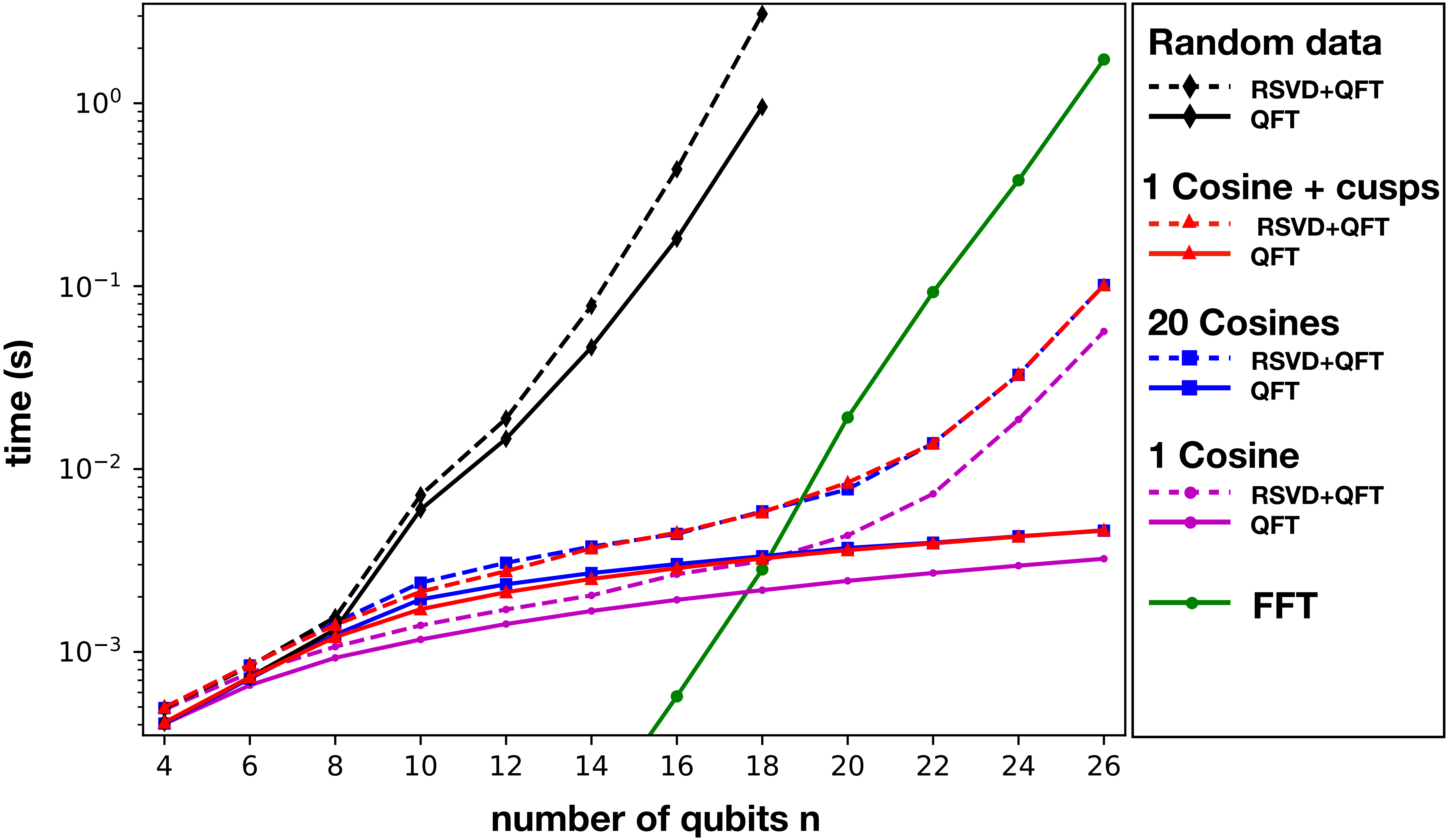


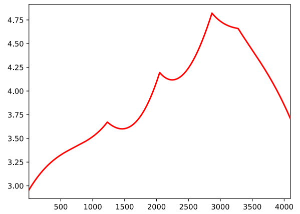
discrete FT using QFT

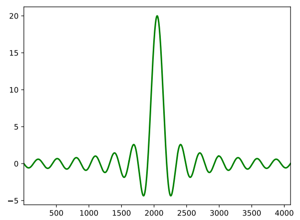


Function Analysis

Performance versus fast Fourier transform (FFT)







Chen, Stoudenmire, White, arxiv:2210.08468

Outlook

ITensor software effort at CCQ not just for quantum problems, but classical too (PDE's, discrete math, etc.)

Tensor network = quantum computer on your laptop

Quickly developing topic – research continuing into:

- more methods of compressing functions into MPS (see next talk)
- improving robustness & efficiency of solvers
- high-performance software
- more general tensor networks beyond MPS (e.g. PEPS)