

Daniel Sharp<sup>1</sup>   Matthew Parno<sup>2</sup>   Michael Brennan<sup>1</sup>   Youssef Marzouk<sup>1</sup>

<sup>1</sup>Massachusetts Institute of Technology

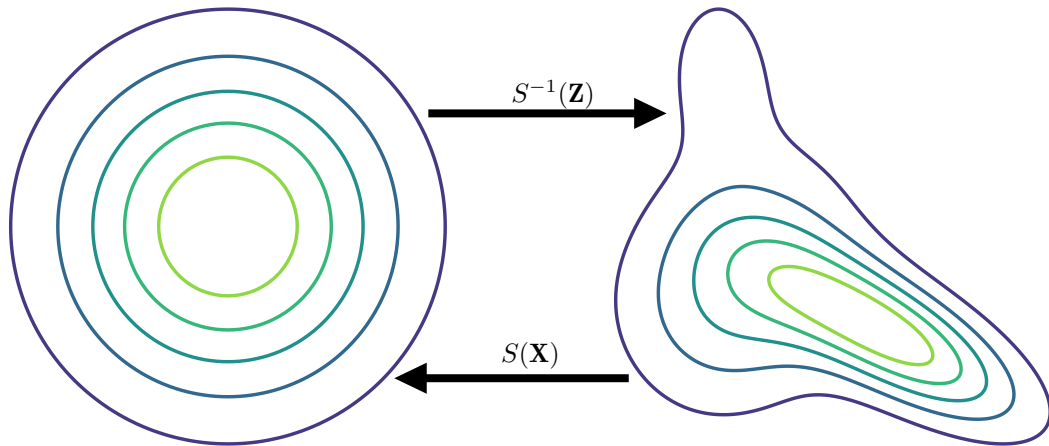
<sup>2</sup>Solea Energy

February 28, 2024



Matthew Parno, Paul-Baptiste Rubio, Daniel Sharp, Michael Brennan, Ricardo Baptista, Henning Bonart, and Youssef Marzouk. “MParT: Monotone Parameterization Toolkit”. In: Journal of Open Source Software 7.80 (2022), p. 4843

# Measure Transport



# Why solve this?

---

- Variational Inference
- Generative Modeling
- Density Estimation
- Data Assimilation
- Conditional Sampling/Simulation-based Inference

# Why solve this?

---

- Variational Inference
- Generative Modeling
- Density Estimation
- Data Assimilation
- Conditional Sampling/Simulation-based Inference

# Why solve this?

---

- Variational Inference
- Generative Modeling
- Density Estimation
- Data Assimilation
- **Conditional Sampling/Simulation-based Inference**

$$\pi_{X|Y} \propto \pi_{Y|X} \pi_X$$

# Problem to Solve

$$S_n(\mathbf{x}) \text{ s.t. } \frac{\partial}{\partial x_n} S_n(\mathbf{x}_{1:n-1}, x_n) > 0$$

$$S(\mathbf{x}) = \begin{bmatrix} S_1(x_1) \\ S_2(x_1, x_2) \\ \vdots \\ S_n(\mathbf{x}_{1:n-1}, x_n) \end{bmatrix}$$

$$\nabla S(\mathbf{x}) = \begin{bmatrix} \partial_1 S_1 & 0 & \cdots & 0 \\ \partial_1 S_2 & \partial_2 S_2 & & 0 \\ \vdots & & \ddots & \vdots \\ \partial_1 S_n & \partial_2 S_n & \cdots & \partial_n S_n \end{bmatrix}$$

# Why triangular? (Computational)

$$S_1(X_1) = Z_1^* \qquad X_1 = S_1(\cdot)^{-1}(Z_1^*)$$

$$S_2(X_1, X_2) = Z_2^* \qquad X_2 = S_2(X_1, \cdot)^{-1}(Z_2^*)$$

$$S_3(X_1, X_2, X_3) = Z_3^* \qquad X_3 = S_3(X_1, X_2, \cdot)^{-1}(Z_3^*)$$

$\vdots$

# Why triangular? (Computational)

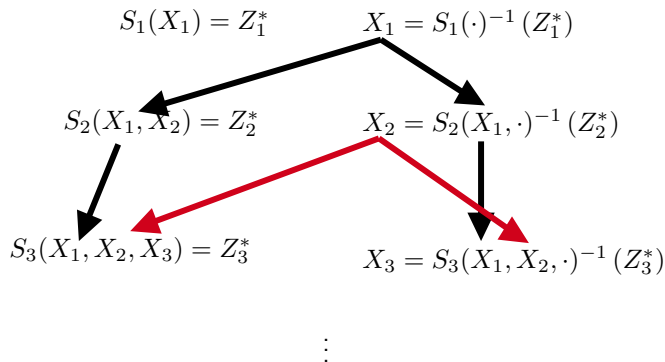
$$\begin{array}{ccc} S_1(X_1) = Z_1^* & X_1 = S_1(\cdot)^{-1}(Z_1^*) & \\ & \swarrow \quad \searrow & \\ S_2(X_1, X_2) = Z_2^* & X_2 = S_2(X_1, \cdot)^{-1}(Z_2^*) & \end{array}$$

$$\begin{array}{ccc} S_3(X_1, X_2, X_3) = Z_3^* & X_3 = S_3(X_1, X_2, \cdot)^{-1}(Z_3^*) & \end{array}$$

$\vdots$



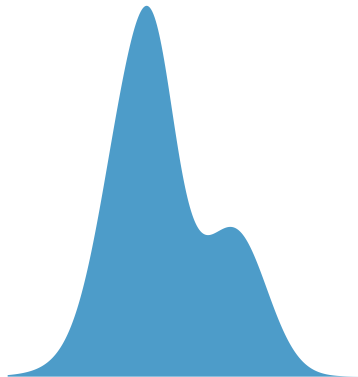
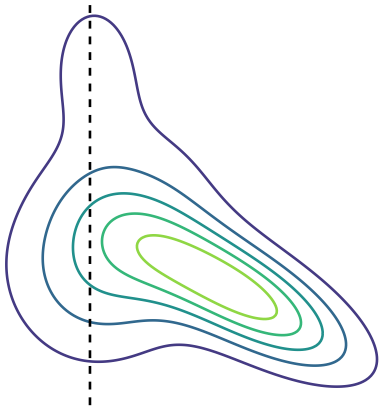
# Why triangular? (Computational)



# Why triangular? (Principle)

$$S_n(\mathbf{X}_{1:n-1}^*, X_n) = Z_n$$

$$S_n(\mathbf{X}_{1:n-1}^*, \cdot)^{-1}(Z_n) = X_n \sim \pi_{X_n|\mathbf{X}_{1:n-1}^*}$$



# What do we want?

---

- Finite training budget (i.e., “Training is not most expensive part”)
- Fast evaluation and training for usage online or in loop-based inference
- Reliable, reproducible results
- Well-understood approximation theory

# What do we want?

---

- Finite training budget (i.e., “Training is not most expensive part”)
- Fast evaluation and training for usage online or in loop-based inference
- Reliable, reproducible results
- Well-understood approximation theory

# What MParT Brings (Software)

- **Fast, parallel, efficient Kokkos-based C++ (on device)**
- **Easy installation with bindings to Python, Matlab, and Julia**
- PyTorch integration
- Out-of-the-box training with NLOpt
- Easy serialization
- Test-driven development, GitHub CI
- Documentation + tutorials

```
$ pip install mpart
$ conda install conda-forge::mpart
julia> ]add MParT
$ cd build && cmake ..
```

# What MParT Brings (Software)

- **Fast, parallel, efficient Kokkos-based C++** (on device)
- **Easy installation with bindings to Python, Matlab, and Julia**
- PyTorch integration
- Out-of-the-box training with NLOpt
- Easy serialization
- Test-driven development, GitHub CI
- Documentation + tutorials

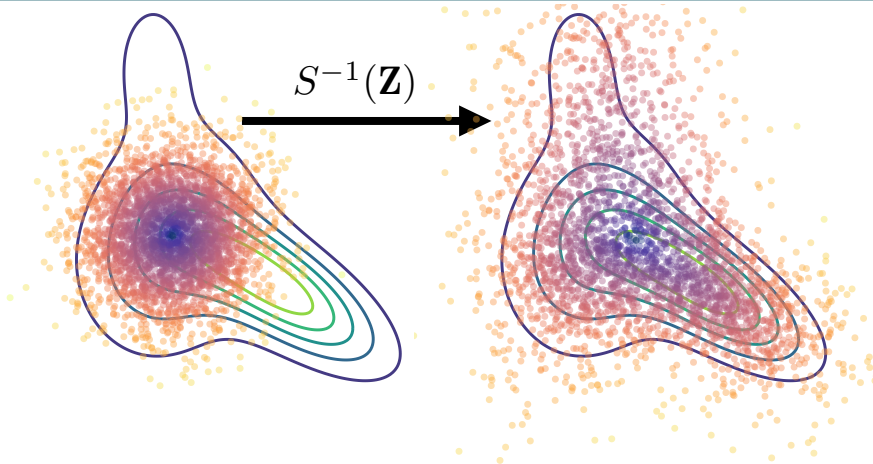
```
$ pip install mpart
$ conda install conda-forge::mpart
julia> ]add MParT
$ cd build && cmake ..
```

# What MParT Brings (Approximation)

---

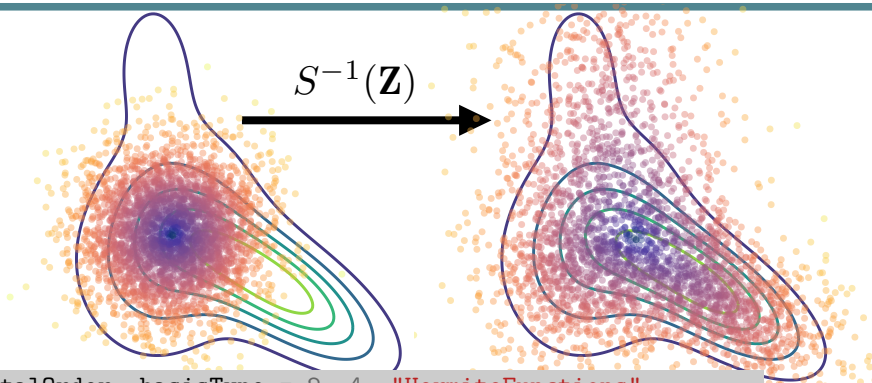
- Fast multivariate expansions and multi-index computation
- Specialized numerical quadrature and root-finding (on device)
- Evaluation, inversion, Jacobian-vector product, parameter gradients...
- Efficient map composition
- Adaptive multi-index set facilities

# Small Example





## Small Example



```
map_dim, totalOrder, basisType = 2, 4, "HermiteFunctions"  
mapOpts = MapOptions(;basisType, basisLB=-4, basisUB=4)  
trimap = CreateTriangular(map_dim, map_dim, totalOrder, mapOpts)  
obj = CreateGaussianKLObjective(samples)  
train_opts = TrainOptions()  
TrainMap(trimap, obj, train_opts)
```

# How to find $S$ ?

---

$$\mathcal{J}(S) = D_{KL}(\pi \| S^\sharp \rho)$$

$$\mathcal{J}_n(S_n) = \sum_{i=1}^N \left[ \frac{1}{2} S_n(\mathbf{x}^{(i)})^2 - \log \partial_n S_n(\mathbf{x}^{(i)}) \right]$$

# What gives us efficiency?

- Rectified integration

$$\log \partial_n S_n(\mathbf{x}) = \sum_{\vec{\alpha}} c_{\vec{\alpha}} \Psi_{\vec{\alpha}}(\mathbf{x}) =: g(\mathbf{x})$$

$$\Rightarrow S_n(\mathbf{x}) = g(\mathbf{x}_{1:n-1}, 0) + \int_0^{x_n} r(g(\mathbf{x}_{1:n-1}, t)) dt, \quad r(\cdot) > 0$$

- Rectified expansion (**NEW**)

$$S_n(\mathbf{x}) = g_1(\mathbf{x}_{1:n-1}) + \sum_{\vec{\beta}} c_{\vec{\beta}} r(\Psi_{\vec{\beta}}(\mathbf{x}_{1:n-1})) s_{\beta_n}(x_n), \quad s'_{\beta_y}(x) > 0$$

# What gives us efficiency?

- Rectified integration

$$\log \partial_n S_n(\mathbf{x}) = \sum_{\vec{\alpha}} c_{\vec{\alpha}} \Psi_{\vec{\alpha}}(\mathbf{x}) =: g(\mathbf{x})$$

$$\Rightarrow S_n(\mathbf{x}) = g(\mathbf{x}_{1:n-1}, 0) + \int_0^{x_n} r(g(\mathbf{x}_{1:n-1}, t)) dt, \quad r(\cdot) > 0$$

- Rectified expansion (**NEW**)

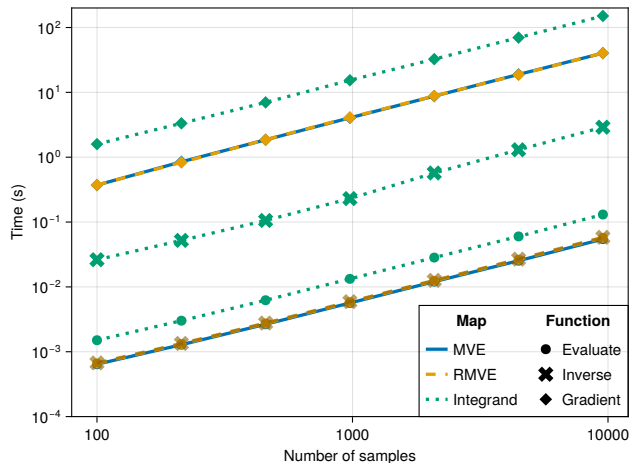
$$S_n(\mathbf{x}) = g_1(\mathbf{x}_{1:n-1}) + \sum_{\vec{\beta}} c_{\vec{\beta}} r(\Psi_{\vec{\beta}}(\mathbf{x}_{1:n-1})) s_{\beta_n}(x_n), \quad s'_{\beta_y}(x) > 0$$

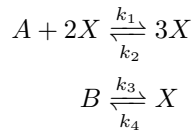
# Scalability

- 1000 dimensions
- Max order 10
- Multi-indices:

$$g(\mathbf{x}) = \sum_{i,j=1}^{d,p} c_{ij} \psi_i(x_j)$$

- **10k parameters, 16 core CPU**

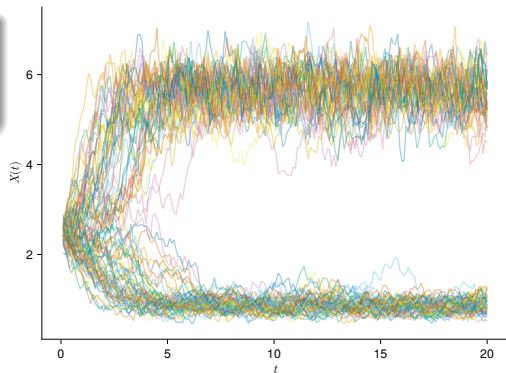




# Sampling a stochastic process

Chemical reaction kinetics

[Sargsyan et al. 09]



# Karhunen Loeve Expansion

$$u(t, \omega) = \mu(t) + \sum_{j=1}^{\infty} \sqrt{\lambda_j} \psi_j(t) X_j(\omega)$$

- $\psi_j$  orthogonal
- $X_j$  uncorrelated, mean 0, variance 1
- We can estimate  $\mu$ ,  $\lambda_j$ ,  $\psi_j$  from samples
- We virtually never know how to sample  $X$ !

# Karhunen Loeve Expansion

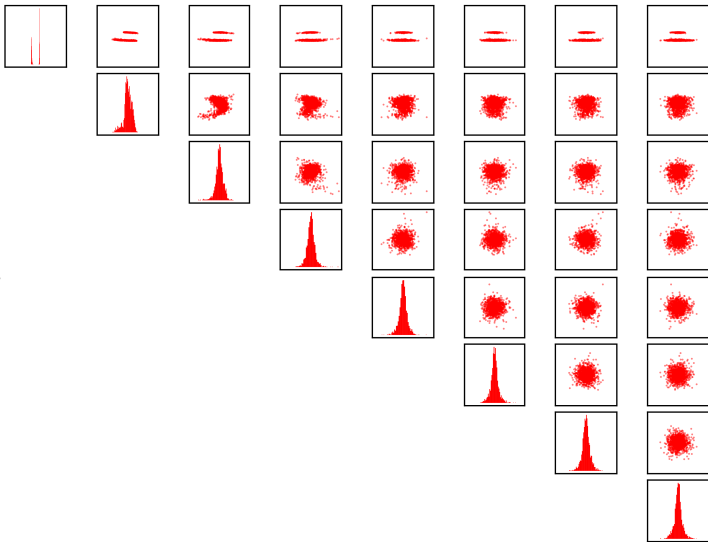
$$u(t, \omega) = \mu(t) + \sum_{j=1}^{\infty} \sqrt{\lambda_j} \psi_j(t) X_j(\omega)$$

- $\psi_j$  orthogonal
- $X_j$  uncorrelated, mean 0, variance 1
- We can estimate  $\mu$ ,  $\lambda_j$ ,  $\psi_j$  from samples
- **We virtually never know how to sample  $X$ !**



# What if...

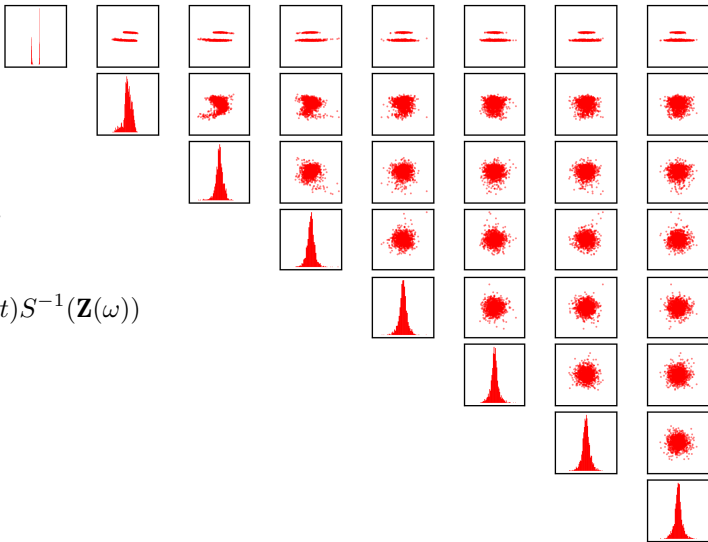
$$\sqrt{\lambda_j} X_j = \langle u - \mu, \psi_j \rangle$$



# What if...

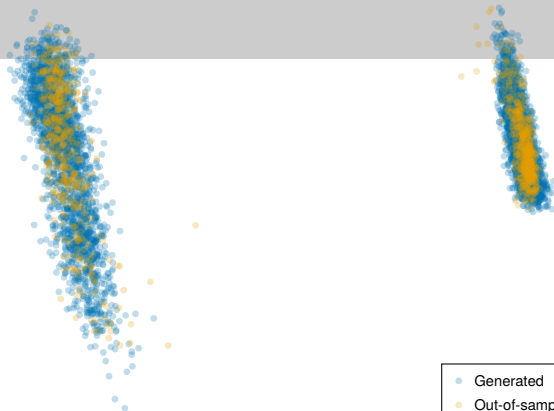
$$\sqrt{\lambda_j} X_j = \langle u - \mu, \psi_j \rangle$$

$$u(t, \omega) \approx \hat{u}(t, \omega) = \mu(t) + \sum_{j=1}^{\infty} \psi_j(t) S^{-1}(\mathbf{Z}(\omega))$$



# Results

```
minOrder, firstOrders=4, [15, 10]
multiindex_sets, opts = KLE_multi_index_setup(dim, minOrder, firstOrders)
map_components = [CreateComponent(mset, opts) for mset in multiindex_sets]
trimap = TriangularMap(map_components)
obj = CreateGaussianKLObjective(samples)
train_opts = TrainOptions()
TrainMap(trimap, obj, train_opts)
```



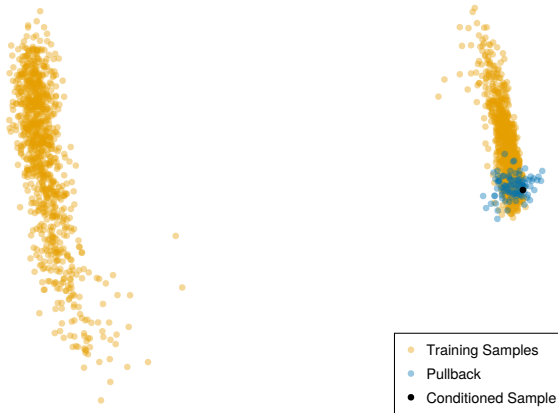
# Conditional Sampling

---

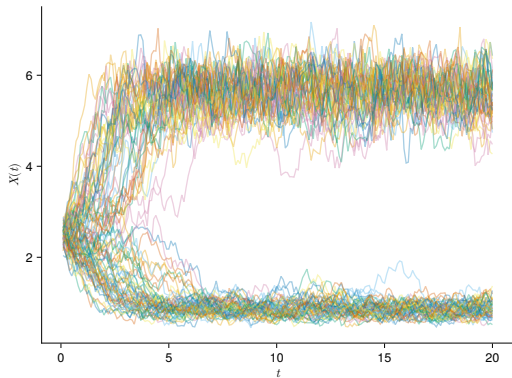
$$S(u(t^*, \omega), X_1, \dots)$$

# Conditional Sampling

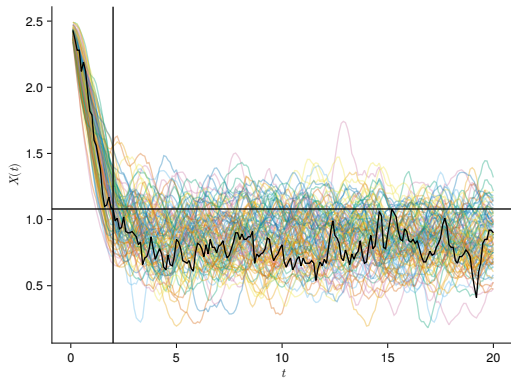
$$S(u(t^*, \omega), X_1, \dots)$$



## Actual Samples



## Conditional, Generated Samples



# Conclusions and Future Work

---

- Scalable to problems yet unseen in triangular transport
  - Improve implementation efficiency further
  - Tailor optimization to parameterizations
  - Improve GPU bindings
- Flexible for many use-cases without sacrificing performance
  - Allow user-specified functions and bases (e.g., Neural-net)
  - Incorporate built-in training for 'map-from-density'
- Easily apply maps or experiment with high-level transport algorithms
  - Incorporate fast quadrature for targets
- All presented materials are available at [dsharp.dev/uq24](https://dsharp.dev/uq24)

# Conclusions and Future Work

---

- Scalable to problems yet unseen in triangular transport
  - Improve implementation efficiency further
  - Tailor optimization to parameterizations
  - Improve GPU bindings
- Flexible for many use-cases without sacrificing performance
  - Allow user-specified functions and bases (e.g., Neural-net)
  - Incorporate built-in training for 'map-from-density'
- Easily apply maps or experiment with high-level transport algorithms
  - Incorporate fast quadrature for targets
- All presented materials are available at [dsharp.dev/uq24](https://dsharp.dev/uq24)



# Conclusions and Future Work

---

- Scalable to problems yet unseen in triangular transport
  - Improve implementation efficiency further
  - Tailor optimization to parameterizations
  - Improve GPU bindings
- Flexible for many use-cases without sacrificing performance
  - Allow user-specified functions and bases (e.g., Neural-net)
  - Incorporate built-in training for 'map-from-density'
- Easily apply maps or experiment with high-level transport algorithms
  - Incorporate fast quadrature for targets
- All presented materials are available at [dsharp.dev/uq24](https://dsharp.dev/uq24)

# References I

- [1] Matthew Parno, Paul-Baptiste Rubio, Daniel Sharp, Michael Brennan, Ricardo Baptista, Henning Bonart, and Youssef Marzouk. “MParT: Monotone Parameterization Toolkit”. In: Journal of Open Source Software 7.80 (2022), p. 4843.
- [2] Youssef Marzouk, Tarek Moselhy, Matthew Parno, and Alessio Spantini. “An introduction to sampling via measure transport”. In: arXiv preprint arXiv:1602.05023 (2016).
- [3] Khachik Sargsyan, Bert Deusschere, Habib Najm, and Olivier Le Matre. “Spectral representation and reduced order modeling of the dynamics of stochastic reaction networks via adaptive data partitioning”. In: SIAM Journal on Scientific Computing 31.6 (2010), pp. 4395–4421.
- [4] Khachik Sargsyan, Bert Deusschere, and Habib Najm. “Spectral Representation and Reduced Order Modeling of Stochastic Reaction Networks”. In: ().
- [5] Fengyi Li et al. “A combinatorial approach to goal-oriented optimal Bayesian experimental design”. PhD thesis. Massachusetts Institute of Technology, 2019.

# Backup Slides (Tailored Optimization)

$$S_n(\mathbf{x}; \mathbf{c}_1, \mathbf{c}_2) := \Psi_1(\mathbf{x}_{1:n-1})\mathbf{c}_1 + f(\mathbf{x}, \mathbf{c}_2)$$

$$\mathcal{J}_n(S_n) = \sum_{i=1}^N \left[ \Psi_1(\mathbf{x}_{1:n-1}^{(i)})\mathbf{c}_1 + f(\mathbf{x}^{(i)}, \mathbf{c}_2) \right]^2 - \mathcal{L}(\mathbf{X}, \mathbf{c}_2)$$

$$= \|\mathbf{A}(\mathbf{X})\mathbf{c}_1 + \mathbf{f}(\mathbf{X}, \mathbf{c}_2)\|^2 - \mathcal{L}(\mathbf{X}, \mathbf{c}_2)$$

$$\hat{\mathbf{c}}_1 = \mathbf{A}^\top \mathbf{A} \mathbf{f}(\mathbf{X}, \hat{\mathbf{c}}_2)$$

$$\hat{\mathbf{c}}_2 = \arg \min_{\mathbf{c}_2} \left\| (\mathbf{A}(\mathbf{A}^\top \mathbf{A})^{-1} + \mathbf{I}) \mathbf{f}(\mathbf{X}, \mathbf{c}_2) \right\|^2 - \mathcal{L}(\mathbf{X}, \mathbf{c}_2)$$

# Backup Slides (Fast Quadrature)

- Assume same Gauss-Hermite quadrature in each dimension,  $\{(z_i, w_i)\}$ .
- Create tensor product grid  $\{\mathbf{z}_{\vec{\alpha}}\}$
- Note  $x_1^{(\vec{\alpha})} = S_1^{-1}(\mathbf{z}_{\vec{\alpha}}) = S_1^{-1}(z_{\alpha_1})$  for any  $\vec{\alpha}$
- Similarly,  $x_2^{(\vec{\alpha})} = S_2(x_1^{(\vec{\alpha})}, \cdot)^{-1}(z_{(\vec{\alpha})})$
- By induction, we can reduce the number of transport map evaluations by an order of magnitude at each dimension.

# Backup Slides (KL Spectrum)

