Variational Monte Carlo with a Neural Network Ansatz to Solve for Groundstate Wavefunctions in Matrix Theory

Korin Aldam-Tajima

April 2025

Abstract

I made a simplified implementation of Han and Hartnoll's deep learning methods for matrix theory. In this report, I approximate the ground state wavefunction $\psi_\theta(X)$ of SU(2) matrix quantum mechanics. Initially, I explore a non-neural approach using a fuzzy sphere radius gaussian ansatz then use a neural network ansatz encoding the wavefunction. A toy neural network approach is achieved by training a deep NN on gauge-fixed matrices to minimize the energy via variational Monte Carlo. I find that this toy approach can approximate the ground wavefunction without the full implementation in Han and Hartnoll. Having a simplified version of Han and Hartnoll's approach is useful for modifications they did not explore. I finally propose a few future extensions of the toy model.

1 Introduction

In quantum gravity, spacetime is not a fixed backdrop but emerges from the entanglement and dynamics of microscopic degrees of freedom. Demonstrating how classical geometries—solutions of Einstein's equations—arise from a fundamental nonperturbative quantum theory remains a key challenge. Matrix quantum mechanics, particularly D0-brane (BFSS/BMN) models, offers a tractable setting in which geometry appears through the eigenvalue dynamics of large matrices.

The objective of this project is to show that deep learning–based variational methods can generate emergent gravitational metrics that closely match the classical Einstein metric in the appropriate regime. This requires evolving density matrices, extracting the ground-state wavefunction, and deriving effective classical equations of motion for collective coordinates—such as a radial "metric" on the fuzzy sphere—and comparing these with predictions from general relativity.

I focus on a bosonic D0-brane Hamiltonian at $N = 2$ (see Sec. 2.1). After gauge-fixing the three 2×2 Hermitian matrices to a reduced set of real parameters, I employ a variational Monte Carlo framework to approximate the ground-state wavefunction $\psi_\theta(X)$. Starting with a simple Gaussian ansatz in the fuzzy-sphere radius, I then adopt a neural network ansatz following Han and Hartnoll's *Deep Quantum Geometry of Matrices* [1]. By training the NN to minimize the Hamiltonian's expectation value, I obtain an efficient sampler for $|\psi_\theta(X)|^2$, which yields accurate variational energies and apparently encodes emergent geometric information.

2 Background

2.1 Matrix Theory

The Hamiltonian

Han and Hartnoll use the following Bosonic-only Hamiltonian [1]:

$$H_B = \mathrm{Tr}\left(\frac{1}{2}\Pi^i\Pi^i - \frac{1}{4}\left[X^i, X^j\right]\left[X^i, X^j\right] + \frac{1}{2}\nu^2 X^i X^i + i\nu\epsilon^{ijk}X^i X^j X^k\right)$$

The quadratic and above terms provide computational complexity which warrant use of stochastic methods which we will explore. The matrices X_i represent the string coordinates of D0-brane matrix theory.

2.2 The Fuzzy Sphere and Initial Gaussian Ansatz

Before I attempt a neural network ansatz, I use a Gaussian ansatz parameterizing the fuzzy sphere radius so that I do not have to deal with the full complexity of the X_i matrices. This way, I can fully implement VMC before moving on to the neural network ansatz. For the fuzzy sphere, it's helpful to think about a point on the two-sphere $S^2 \subset \mathbb{R}^3$ satisfies

$$r^2 \;=\; x_1^2 + x_2^2 + x_3^2,$$

so that the locus $r = \mathrm{const}$ is the ordinary sphere of radius r.

In the *fuzzy-sphere* construction, the coordinate functions x_i are replaced by three Hermitian matrices X_i, and one defines the "radial operator"

$$\widehat{r}^2 \;=\; X_1^2 + X_2^2 + X_3^2.$$

A natural scalar radius is then obtained by taking the (normalized) trace:

$$r(X) = \sqrt{\frac{\mathrm{Tr}\big(X_1^2 + X_2^2 + X_3^2\big)}{N}}\,,$$

where $N = \dim(X_i)$. In our toy model we set $N = 2$, so

$$r(X) = \sqrt{\frac{\mathrm{Tr}(X_1^2 + X_2^2 + X_3^2)}{2}}\,.$$

In our Variational Monte Carlo toy model we compute

$$r(X) \;=\; \sqrt{\frac{\Re\,\mathrm{Tr}(X_1^2 + X_2^2 + X_3^2)}{2}}$$

for each sample X, and treat r as the one-dimensional coordinate in our Gaussian ansatz $\psi(r) \propto \exp[-(r - r_0)^2/(2\sigma^2)]$.

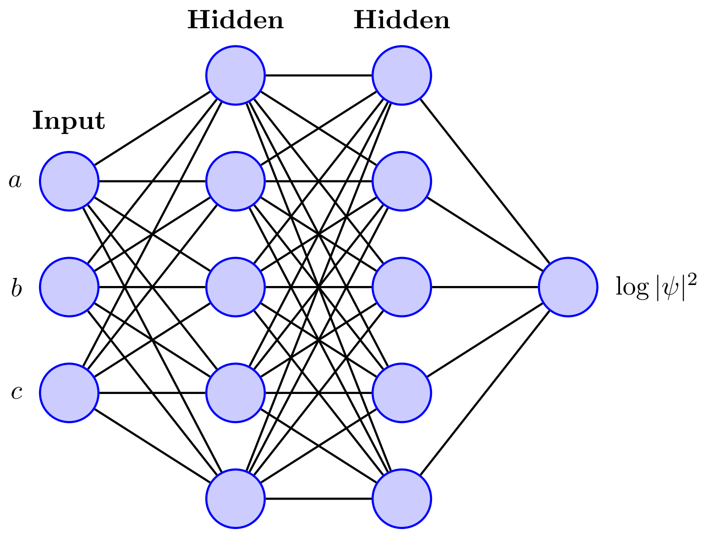


Figure 1: Neural-network ansatz schematic: 3-dimensional input (a, b, c), two hidden layers of size 64 with tanh activations, and a single scalar output $\log|\psi|^2$.

2.3 Neural-Network Wavefunction Ansatz

The neural network is trained to minimize the expectation energy via variational Monte Carlo, allowing efficient sampling and approximation of the ground state for SU(N) matrix quantum mechanics at large N. In Deep Quantum Geometry of Matrices, Han and Hartnoll approximate the ground state wavefunction $\psi_\theta(X)$ using deep generative flows, specifically normalizing flows (NF) and masked autoregressive flows (MAF). While these are interesting and probably very useful, I will explore a toy model using more tools learned in 170X and base the project on the core ideas of using VMC and a neural network ansatz for the wavefunction.

We replace the simple two-parameter Gaussian by a small multilayer perceptron (MLP) whose weights serve as variational parameters. After diagonalizing and phase-fixing the matrices, each configuration $X = (X_1, X_2, X_3)$ is uniquely represented by three real numbers

$$a = \Re\big(X_1^{11}\big), \quad b = \Re\big(X_1^{22}\big), \quad c = \Im\big(X_2^{12}\big).$$

We collect these into the input vector

$$x = (a, b, c) \;\in\; \mathbb{R}^3.$$

Let θ denote the full set of weights and biases of an MLP. Here implemented as (see Figure 1)

$$\texttt{Chain(Dense(}3,\ 64,\ \tanh), \ldots, \mathrm{Dense}(64, 1))$$

. We then define

$$f_\theta(x) \;\approx\; \log\big|\psi_\theta(X)\big|^2$$

up to an additive constant. The variational wavefunction is given by

$$|\psi_\theta(X)|^2 \;=\; \exp\big(f_\theta(x)\big),$$

so that

$$\log |\psi_\theta(X)|^2 = f_\theta(x).$$

Each individual weight and bias in θ is treated as a variational parameter, analogous to r_0 and σ in the Gaussian toy. Together, they parameterize a function $f_\theta : \mathbb{R}^3 \to \mathbb{R}$, capable of capturing multi-modal, anisotropic, and phase-correlated features of the true ground-state wavefunction. During training, gradient-based optimization adjusts θ to minimize the variational energy $\langle H \rangle = \int dX \, |\psi_\theta(X)|^2 \, E_{\text{loc}}(X)$, thereby "sculpting" $|\psi_\theta(X)|^2$ to approximate the exact ground-state distribution.

3 Pluto Notebook Implementation

The code for this project can be found at `https://github.com/THENIROCK/VMC_NN_Groundstate_Estimation`. Here, I will cover each cell in the two Pluto notebooks. The notebooks are mostly the same except one uses a Gaussian ansatz and the other uses a neural network ansatz.

Package Imports

The first cell loads the Julia packages required for interactive computation, automatic differentiation, random sampling, linear algebra, and plotting:

- `PlutoUI` for interactive sliders and bindings.
- `Flux` for gradient-based optimization and AD.
- `Distributions` and `Random` for sampling from normal distributions.
- `LinearAlgebra` for matrix operations and traces.
- `Plots` for visualizing results.
- `Functors` to allow Flux to access the variational parameters.

Pauli Matrices and Generators

This cell defines the standard Pauli matrices $\sigma_x, \sigma_y, \sigma_z$ and constructs the corresponding simple unitary $SU(2)$ generators $J_i = \sigma_i/2$. These 2×2 Hermitian matrices satisfy the commutation relations

$$[J_i, J_j] = i\epsilon_{ijk} J_k,$$

and serve as a basis for the bosonic matrix degrees of freedom in the minimal BMN truncation.

`HB`: Bosonic mini-BMN Hamiltonian

I Implement the purely bosonic Hamiltonian for the mass-deformed BMN model at $N = 2$. Below is an explicit form, derived in Appendix A, of the bosonic Hamiltonian introduced in section 2.1. In matrix notation,

$$H_B(X_1, X_2, X_3; \nu) = \frac{1}{2}\,\mathrm{Tr}\big([X_i, X_j]^2\big) + \frac{1}{2}\nu^2\,\mathrm{Tr}(X_i^2) + \nu\,\Im\,\mathrm{Tr}\big(X_1X_2X_3 - X_1X_3X_2\big) \tag{1}$$

where X_i are 2×2 Hermitian matrices and ν is the mass-deformation parameter. Commutators and traces are evaluated explicitly on dense matrices to ensure compatibility with automatic differentiation with the gauge-fixed matrices

$$X_1 = \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}, \qquad X_2 = \begin{pmatrix} 0 & i\,c \\ -i\,c & 0 \end{pmatrix}, \qquad X_3 = \begin{pmatrix} d & 0 \\ 0 & d \end{pmatrix} \text{ or } 0_{2\times 2}$$

. Here, a, b, c, d parameterize either the gaussian ansatz or the neural network wavefunction ansatz and are Monte Carlo sampled.

`Ansatz`: Gaussian Ansatz and Wavefunction Probability

I first performed VMC without a neural network ansatz. I introduce a two-parameter Gaussian ansatz on the "fuzzy-sphere" radius of the configuration $X = (X_1, X_2, X_3)$ instead of the string coordinate matrices X_i.

Use the fuzzy sphere radius to compress the matrices into a more tractable ansatz (since we aren't using a neural network yet).

$$r(X) = \sqrt{\frac{\Re\ \mathrm{Tr}\big(X_1^2 + X_2^2 + X_3^2\big)}{2}}\,,$$

so that $r(X)$ measures the root-mean-square size of the three matrices.

Gaussian ansatz Let $\theta = (r_0, \sigma)$ denote the variational parameters. We choose

$$|\psi_\theta(X)|^2 \ \propto\ \exp\Big[-\frac{\big(r(X) - r_0\big)^2}{\sigma^2}\Big].$$

Equivalently, defining the log-probability function,

$$\log|\psi_\theta(X)|^2 \,=\, -\,\frac{\big(r(X) - r_0\big)^2}{\sigma^2}\,,$$

which in code corresponds to

```
function radius(X1,X2,X3)
    return sqrt(real(tr(X1^2+X2^2+X3^2))/2)
end

function logp(ans::Ansatz, X1,X2,X3)
    r = radius(X1,X2,X3)
    return -((r - ans.r0)^2)/(ans.sigma^2)
end
```

Here `Ansatz(r0,`σ`)` packages the parameters (r_0, σ). The function `logp` returns $\log|\psi|^2$, ready for use in Monte Carlo sampling and gradient-based optimization.

Neural Network Ansatz

Next I replaced the gaussian wavefunction ansatz with a neural network. Now we can handle the string coordinte matrices X^i more directly. The struct `Ansatz`: holds a `Chain(3→hidden→hidden→1)` dense neural network (see Figure 1). I used a hidden layer of dimension 64. This MLP is initialized with tanh layers. Han and Hartnoll use tanh but also state that the activation function didn't really impact their results. I use tanh because they did.

`radius`, `logp`: Mapping Ansatz to Probability

This cell defines the functions `radius` and `logp`, which compute the fuzzy-sphere radius $r(X)$ and the logarithm of the probability density $\log|\psi(X)|^2$, respectively. These establish the connection between the variational parameters (r_0, σ) and the weight used for Monte Carlo sampling.

`energy`: Monte Carlo Estimator for $\langle H \rangle$

Here the function `energy(ans,`ν`;nsamples)` carries out a simple gauge-fixed Monte Carlo integration of the variational energy

$$E[\psi] = \frac{\int dX\, |\psi(X)|^2\, H_B(X)}{\int dX\, |\psi(X)|^2}$$

by sampling n_{samples} configurations (diagonal X_1 and one off-diagonal mode in X_2), weighting each by $|\psi(X)|^2$, and averaging H_B. See Appendix B for the gauge-fixing and simplifications made. The estimator is unbiased but carries sampling noise.

`loss`, Optimization Loop: Variational Optimization

Define a loss function `loss(ansatz)` that wraps `energy`, varying $\theta = (r_0, \sigma)$or (a, b, c, d). The parameter linear mass deformation parameter d is optional and I ended up ignoring it because it caused problems. Flux's automatic differentiation computes

$$\nabla_\theta E$$

, and a simple gradient-descent loop updates these two variational parameters over multiple epochs. An interactive slider for ν allows exploration of how the optimal radius depends on the mass deformation.

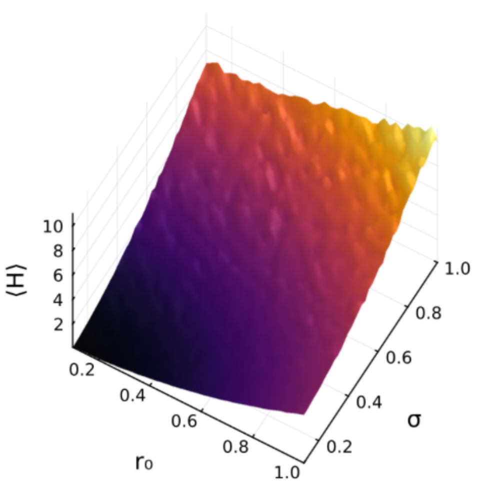
Energy vs. r_0 Visualization

Compute $\langle H \rangle$ across a range of r_0 values at fixed ν and plot the resulting curve using `plot`. This visualization confirms the variational principle by locating the minimum of $E(r_0)$.

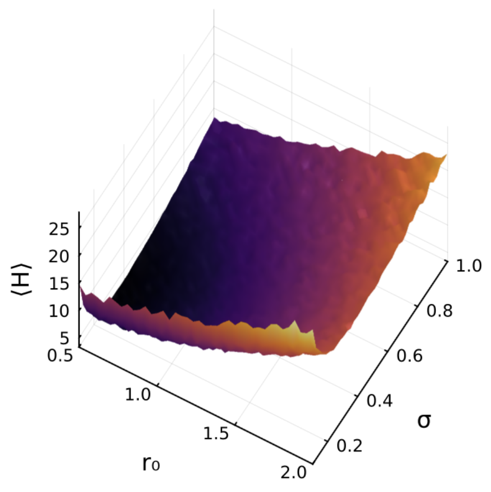
4 Results and Verification

For the Gaussian ansatz model with $X3 = 0_{2\times 2}$, I obtained

$$r_0 = 0.02, \qquad \sigma = 0.667, \qquad \langle H \rangle \approx 3.5$$



(a) Gaussian ansatz Monte Carlo energy landscape (no kinetic term), minimum $r_0 = 0, \sigma = 0$.



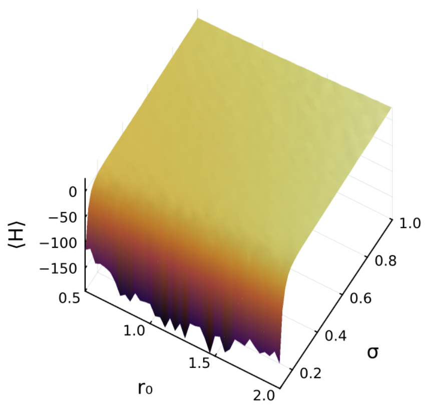
(b) Full Gaussian ansatz Monte Carlo energy landscape, minimum $r_0 = 0.02, \sigma = 0.667$.

Figure 2: Monte Carlo energy landscapes. $\langle H \rangle \approx 3.5$

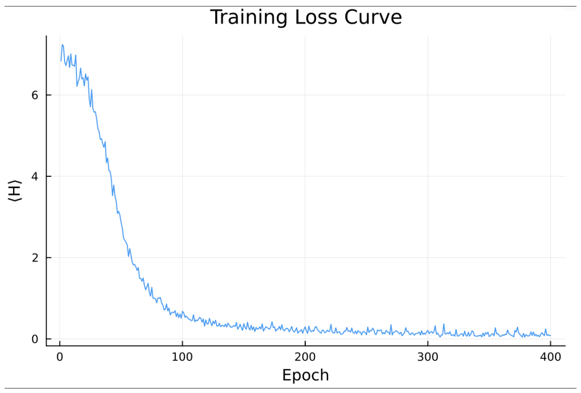
. The Monte-Carlo-sampled energy landscape appears to be increasing uniformly in both r_0 and σ. However, there is a small lip near the origin which leads to non-zero values for r_0 and σ (see Figure 2b). To verify the algorithm, I removed the kinetic term from the Hamiltonian to see if we would get a delta function at the center of the potential. Indeed, this seems to be the case (see Figure 2a) which adds confidence to my code.

Increasing complexity (still using Guassian ansatz), I added $X_3 = \begin{pmatrix} d & 0 \\ 0 & d \end{pmatrix}$ to include non-zero linear mass deformation ν effects. This didn't work so well which is why I initially removed it. We get a long valley (see Figure 3a) and NaN values for r_0 and σ which I suspect is due to the more sophisticated gauge fixing procedures used by Han and Hartnoll which I ignored for this toy model.

Finally, the neural network ansatz minimized to a ground state energy of $\langle H \rangle \approx 0.56$. A whole 3 energy units lower than the gaussian ansatz method $\langle H \rangle \approx 3.5$ which is very promising. Visualizing what this wavefunction looks like is hard as it is described in terms of weights in a neural network. I *can* graph the training loss (see Figure 3b) over epochs but it isn't very illuminating. You may notice that compared to the most commonly seen MLP training graphs, this looks more like a sigmoid than an exponential decay which I think is because we used **tanh** as the activation function.



(a) Linear mass deformation Gaussian ansatz Monte Carlo energy landscape, minimum $r_0 = \text{NaN}, \sigma = \text{NaN}$.



(b) Loss of neural network ansatz architecture. Final $\langle H \rangle = 0.56$

Figure 3: Linear mass deformation energy landscape and neural network ansatz training loss.

5 Conclusion and Outlook

This was also a highly simplified toy model – simplified to the point where I was just trying to get the code to work. The cubic term of the Hamiltonian was eliminated which encodes the linear effects of the mass deformation ν. For real-world use, a thoughtfully put together Hamiltonian for code should be used. As a proof of concept, though, the neural network ansatz and Variational Monte Carlo method for ground state wavefunction estimation in Matrix Theory appears promising.

For the future, we can obviously tune hyper-parameters and change the activation function from tanh to capture various features of a wavefunction better. I was also thinking running the Process in reverse by starting with real data and training a neural net to predict the Hamiltonian or ground state wavefunction parameters would be useful to match the theory with observations. Data could be the Einstein ring etc. data from the Euclid Q1 data release. Including fermionic terms to visualize gravitation, or some dimensionality reduction using t-SNE/PCA to visualize the fuzzy sphere of D0-branes would be fun extensions.

6 Acknowledgements

Thank you to Prof Sahakian for teaching some awesome 170x machine learning techniques and covering some D0-brane string theory that enabled me to approximately understand what (on earth) I am doing.

References

[1] Xizhi Han and Sean A. Hartnoll. "Deep Quantum Geometry of Matrices". In: *Physical Review X* 10.1 (Mar. 2020). ISSN: 2160-3308. DOI: `10.1103/physrevx.10.011069`. URL: `http://dx.doi.org/10.1103/PhysRevX.10.011069`.

[2] Enrico Rinaldi et al. "Matrix-Model Simulations Using Quantum Computing, Deep Learning, and Lattice Monte Carlo". In: *PRX Quantum* 3.1 (Feb. 2022). ISSN: 2691-3399. DOI: `10.1103/prxquantum.3.010324`. URL: `http://dx.doi.org/10.1103/PRXQuantum.3.010324`.

A Code-implementable form of Bosonic Hamiltonian

The last term in the Hamiltonian is confusing. Here's where it comes from. Start with Han and Hartnoll's Hamiltonian:

$$H_B = \mathrm{Tr}\left(\frac{1}{2}\Pi^i\Pi^i - \frac{1}{4}\left[X^i, X^j\right]\left[X^i, X^j\right] + \frac{1}{2}\nu^2 X^i X^i + i\nu\epsilon^{ijk} X^i X^j X^k\right)$$

Using the property

$$\mathrm{Tr}(ABC) = \mathrm{Tr}(CAB) = \mathrm{Tr}(BCA)$$

we can explictly expand the last term into a cyclic and non-cyclic part

$$i\nu\epsilon^{ijk} X^i X^j X^k = 3i(X^1X^2X^3 - X^3X^2X^1)$$

which is implemented in the code as

```
Vs = v * imag(tr(A*B*C-C*B*A))
```

B Gauge Fixing in the 2×2 Bosonic Mini-BMN Toy Model

I just let ChatGPT do this part. The bosonic mini-BMN model at $N = 2$ has three 2×2 Hermitian matrices X_1, X_2, X_3 with

$$X_i \longmapsto U\, X_i\, U^\dagger \quad , \qquad U \in SU(2)$$

gauge symmetry. We gauge-fix down to three real parameters (a, b, c) as follows:

1. **Diagonalize X_1.** Use an $SU(2)$ rotation to set

$$X_1 \;=\; U\, X_1^{(\mathrm{orig})}\, U^\dagger \;\longrightarrow\; \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}, \quad a, b \in \mathbb{R}.$$

   This fixes two of the three gauge parameters.

2. **Fix the residual $U(1)$ phase.** The subgroup commuting with $\mathrm{diag}(a, b)$ is $\{\mathrm{diag}(e^{i\phi}, e^{-i\phi})\}$. Use this to rotate the off-diagonal phase of X_2 so that

$$X_2 \;=\; \begin{pmatrix} 0 & i\,c \\ -i\,c & 0 \end{pmatrix}, \quad c \in \mathbb{R}.$$

   This fixes the last continuous gauge direction.

3. **Truncate X_3.** For simplicity in the NN toy model we set

$$X_3 \;=\; 0_{2\times 2},$$

rather than sampling its remaining four real entries. The Gaussian ansatz model explored this as a non-zero term.

4. **Resulting parameterization.** Instead of sampling all twelve real entries of (X_1, X_2, X_3) (with redundant gauge orbits), we sample

$$(a, b, c) \;\in\; \mathbb{R}^3$$

and reconstruct

$$X_1 = \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}, \quad X_2 = \begin{pmatrix} 0 & i\,c \\ -i\,c & 0 \end{pmatrix}, \quad X_3 = 0.$$

5. **Note on the Vandermonde Jacobian.** A fully consistent gauge-fixing would introduce the factor $\prod_{i<j}(a_i - a_j)^2 = (a-b)^2$ in the integration measure. In this toy implementation we ignore this Jacobian and treat sampling in (a, b, c) as a proof of concept.

With this gauge choice, the Monte Carlo loop only draws

$$a = r_0 + \sigma\, z_1, \quad b = r_0 + \sigma\, z_2, \quad c = \sigma\, z_3, \quad z_i \sim \mathcal{N}(0,1),$$

and builds the Hamiltonian

$$H_B(X_1, X_2, X_3; \nu) = \tfrac{1}{2}\sum_{i<j} \mathrm{tr}\big([X_i, X_j]^2\big) + \tfrac{1}{2}\,\nu^2 \sum_i \mathrm{tr}(X_i^2) + \nu\,\Im\,\mathrm{tr}(X_1X_2X_3 - X_1X_3X_2)$$

on these three matrices.