Understanding DQM

A Technical Summary of Dynamic Quantum Mapping, Covering the Algorithm and its Implementation

Zander Teller

April 24, 2023

Contents

1 Purpose and Prerequisites

This paper is intended to enable a detailed understanding of:

- the conceptual underpinnings of the DQM algorithm
- how the DQM algorithm is actually implemented

This paper does *not* cover how to use DQM or how to interpret DQM results. (See the *DQM User Guide* for these topics.)

Basic knowledge is assumed on various subjects, including complex numbers, linear algebra, Hilbert spaces, and some of the basics of quantum mechanics (e.g., Dirac bra-ket notation, and the concept of a quantum potential).

2 Dynamic Quantum Mapping

Dynamic Quantum Mapping (DQM) is an algorithm designed for exploring and understanding the intrinsic structure of a data set. The starting point is always a set of n points in some d-dimensional Euclidean data space. DQM begins by building a high-dimensional landscape of data density, where denser regions are lower and less dense regions are higher in the landscape. It then moves each data point downhill within that landscape. Points that gather together reveal clusters, and animated visualization of the points' motion can further reveal extended structures (e.g., lines, like streambeds) within the landscape.

The power of DQM comes from the fact that, if data points lie near some lower-dimensional manifold within the d-dimensional space, the points will move toward that manifold, revealing its structure more clearly. Any such manifolds are determined entirely by the location of the data points within the space; there are no other limiting assumptions about what structures may be discovered.

Dynamic

As the data points move, information is contained in their journey as well as in their final destination. In the landscape metaphor: if points flow like water droplets along a 'streambed' to gather in a 'lake', the streambed may be as important as the lake in understanding the structure of the data set.

The fact that visualizations are dynamic (i.e., animated) is therefore also important; the motion allows a single animated 3-dimensional plot to reveal structure in more than 3 dimensions (since the motion is driven by information from all dimensions, not just the 3 in the visualization).

Quantum

The DQM algorithm borrows heavily from the mathematical framework of quantum mechanics. Crucially, the quantum-mechanical framework provides non-local gradient descent and the phenomenon of tunneling. The landscape created by a set of data points will generally be non-convex, which creates a serious problem for classical methods of gradient descent. Quantum tunneling allows a point to ignore or pass through minor potential barriers (i.e., small 'hills') on its way to some nearby lower region. DQM in this way reveals the structure of a data landscape while ignoring the smallest uninteresting details.

Mapping

DQM is a clustering algorithm[1], where points gather together and so reveal clusters in the data. However, it is also more than a clustering algorithm. DQM can reveal extended structures within a data set, both 1-dimensional 'streambed' structures and higher-dimensional manifolds. In the most general sense, these extended structures can be thought of as regressions, since they reveal parameterized relationships between the different dimensions of the data space. Extended structures can also operate as subclusters; for example, in the landscape metaphor, two streams feeding into the same lake from different directions can be treated as two subclusters of the same cluster.

3 DQM Technical Summary

Given a set of n data points in a d-dimensional Euclidean space, DQM begins by putting a d-dimensional Gaussian distribution around each data point; the Gaussian for a given point is considered to be that point's initial quantum wave function. DQM then adds all of these Gaussians together, and this sum is treated as the quantum wave function for the entire data set. (Think of the sum of Gaussians as a generalized representation of where there are likely to be data points in the space.)

DQM uses the time-independent Schrödinger wave equation to define a quantum potential for which the data set's overall wave function is a stable solution; this potential is the 'landscape' defined by the data set. DQM then evolves each data point's individual wave function (the initial d-dimensional Gaussian) in this potential using the time-dependent Schrödinger wave equation. At each time step, each point's new position is calculated from the expected position of its evolved wave function. The sequence of positions creates a trajectory for each point through the data space as it moves downhill in the DQM landscape[2]. Evolution can continue until all points have reached some minimum in the potential and stopped moving.

[1] The algorithm's original name was Dynamic Quantum Clustering (DQC). The name was changed to reflect the fact that clustering is only a part of what DQM can do.

[2] Ehrenfest's Theorem from quantum mechanics is how we know that quantum evolution does, in fact, result in a point moving downhill in the landscape. (See Appendix A.)

4 Initial Wave Functions

In this section, we introduce several important building blocks:

- the idea of a quantum wave function
- the initial Gaussian wave function for each data point
- the overall wave function for the data set
- calculation of inner products ('overlaps') between Gaussian wave functions

4.1 Introduction to Quantum Wave Functions

In a data set with d real-valued dimensions ($\mathbb{R}^d$), the jth data point exists at a position $\vec{x}_j$:

$$\vec{x}_j \;=\; < x_1,\, x_2,\, ...\; x_d > \tag{4.1}$$

A *quantum wave function* $\psi_j(\vec{x})$ describing this data point is a complex-valued function with a value at every point in the $\mathbb{R}^d$ space. The wave function can be loosely interpreted as a generalized notion of 'where the point is'. At an arbitrary point in space $\vec{x}_0$, the complex value $\psi_j(\vec{x}_0)$ is interpretable as a *probability amplitude*, and its squared norm is interpretable as a *probability density* ρ_0:

$$\rho_0 = |\psi_j(\vec{x}_0)|^2 = \; \psi_j(\vec{x}_0)^* \psi_j(\vec{x}_0) \tag{4.2}$$

where $\psi_j(\vec{x}_0)^*$ is the complex conjugate of $\psi_j(\vec{x}_0)$.

The wave function is *normalized* if it meets the normalization criterion:

$$\int_{-\infty}^{\infty} \rho(\vec{x})\, d\vec{x} \;=\; \int_{-\infty}^{\infty} |\psi_j(\vec{x})|^2\, d\vec{x} \;=\; \int_{-\infty}^{\infty} \psi_j^*(\vec{x})\, \psi_j(\vec{x})\, d\vec{x} \;=\; 1 \tag{4.3}$$

which means that the probability of finding the point somewhere in space is exactly 1.

4.2 Initial Gaussian Wave Functions in DQM

In DQM, the jth data point is initially represented by a normalized d-dimensional Gaussian wave function centered at the data point's initial position $\vec{x}_j$:

$$\psi_j(\vec{x}) \;=\; C_g\, e^{-\frac{|\vec{x}-\vec{x}_j|^2}{2\sigma^2}} \tag{4.4}$$

The normalization criterion (Equation 4.3) gives us:

$$C_g = (\pi\sigma^2)^{-\frac{d}{4}} \tag{4.5}$$

In practice, C_g cancels out in every DQM calculation where it might appear, and so we are never concerned with its precise value.

The width of the Gaussian, σ, is the same in every dimension. (Understood as a multivariate Gaussian, the wave function has a diagonal covariance matrix, with variances of σ^2 everywhere along the diagonal.) The wave function for every data point will have the same value of σ. Thus, σ is DQM's single main tunable parameter.

DQM itself does not specify how to choose σ. For any data set, the extremes are always the same: for sufficiently small σ, points will not move at all (every point has its own well in the landscape); for sufficiently large σ, all points will come together directly into a single point cluster (all points are inside a single well in the landscape). Exploring intermediate values of σ will reveal the inherent structure of the data set. (For more on this topic, see the *DQM User Guide*.)

4.3 Overall Wave Function for the Data Set

The overall wave function for the full, entire data set is the sum of the initial Gaussian wave functions for each data point:

$$\psi_f(\vec{x}) \;=\; \sum_{j=1}^{n} C_g \, e^{-\frac{|\vec{x}-\vec{x}_j|^2}{2\sigma^2}} \;=\; C_g \sum_{j=1}^{n} e^{-\frac{|\vec{x}-\vec{x}_j|^2}{2\sigma^2}} \tag{4.6}$$

(The 'f' subscript is for 'full'.) This overall wave function is used in building the *quantum potential* (see Section 6.5). It can be loosely interpreted as a description of where there are likely to be data points in the space, but we never need to worry about normalization of the overall wave function (see Section 6.5.1).

At this point, the data set's overall wave function ψ_f looks a lot like the Parzen-Rosenblatt window estimator, but the similarity ends when we treat it as a wave function within the mathematical framework of quantum mechanics.

4.4 Inner Product of Gaussian Wave Functions

The inner product of any two of these initial Gaussian wave functions is:

$$\langle \psi_i \,|\, \psi_j \rangle \;=\; \int_{-\infty}^{\infty} \psi_i^*(\vec{x}) \, \psi_j(\vec{x}) \, d\vec{x} \;=\; {C_g}^2 \int_{-\infty}^{\infty} e^{-\frac{|\vec{x}-\vec{x}_i|^2}{2\sigma^2}} e^{-\frac{|\vec{x}-\vec{x}_j|^2}{2\sigma^2}} \, d\vec{x} \tag{4.7}$$

We will generally refer to this inner product as the *overlap* between any two wave functions.

Algebraic manipulation of the exponents and the normalization criterion allow us to simplify Equation 4.7:

$$\langle \psi_i \,|\, \psi_j \rangle \;=\; {C_g}^2 \int_{-\infty}^{\infty} e^{-\frac{|\vec{x}-\vec{x}_j|^2}{2\sigma^2}} e^{-\frac{|\vec{x}-\vec{x}_j|^2}{2\sigma^2}}\, d\vec{x} \;=$$

$$e^{-\frac{|\vec{x}_i-\vec{x}_j|^2}{4\sigma^2}} \Big({C_g}^2 \int_{-\infty}^{\infty} e^{-\frac{|\vec{x}-\frac{\vec{x}_i+\vec{x}_j}{2}|^2}{2\sigma^2}} e^{-\frac{|\vec{x}-\frac{\vec{x}_i+\vec{x}_j}{2}|^2}{2\sigma^2}}\, d\vec{x} \Big) \;=\; e^{-\frac{|\vec{x}_i-\vec{x}_j|^2}{4\sigma^2}} \tag{4.8}$$

The Gaussians are normalized, so we always have $\langle \psi_j \,|\, \psi_j \rangle = |\psi_j|^2 = 1$. Equation 4.8 correctly produces this result for $i = j$.

We can think of two Gaussian wave functions as being *essentially linearly independent* if their overlap is very small, which happens in Equation 4.8 when their centers are well separated relative to the value of σ. They're close to orthogonal in this case, since their overlap (inner product) is close to zero. This idea becomes important in creating a basis for the data set (see Section 5.2).

5 Creating the Basis

This section deals with the mechanics of representing an arbitrary wave function in a computationally tractable way.

5.1 General Representation of Wave Functions

All possible normalized wave functions exist in the infinite-dimensional Hilbert space of square-normalized complex-valued functions in $\mathbb{R}^d$. ('Square-normalized' is just a restatement of the normalization criterion in Equation 4.3). The initial Gaussian wave function for each data point will evolve under the influence of the quantum potential (see Section 7.3), and so we need a computationally tractable way to represent general wave functions that are no longer simple Gaussians. DQM does this by choosing a set of *orthonormal quantum eigenstates*; an arbitrary wave function will then be represented (usually imperfectly) as a linear combination of these eigenstates.

5.2 Choosing a Set of Basis Points

First, we choose a subset of the n data-set points, from which the quantum eigenstates will be built. For small n, we can use all points from the data set as the basis points. For larger n, we need to choose a subset of the data points, for computational tractability. The size limit for the basis depends on available computing resources[3]. The size of the basis is analogous to a 'resolution' for the landscape that DQM constructs for a given data set; a larger basis yields a more detailed landscape but is more computationally expensive.

[3]Using a single 32-core 3.7GHz processor, execution speed typically starts to become painfully slow somewhere between 1,000 and 2,000 basis points.

When choosing a subset of points for the basis, we want the basis points to be as close to linearly independent as possible, which means we want them to be as far away from each other as possible. (The original DQC paper[4] refers to this process as choosing the "maximally essentially linearly independent states".) This selection of basis points can be done in multiple ways. One approach is as follows: choose the first data point for the basis at random (or start with the biggest outlier). Then, iteratively add to the basis the non-basis point whose closest distance to any current basis point is largest, until the desired basis size is reached.

5.3 Constructing the Basis Eigenstates

Any wave function can be described by a vector of its overlaps with each of the basis-point Gaussian wave functions. In a loose sense, this vector of overlaps can be thought of as the 'coordinates' of the wave function in the basis. This idea may be helpful in understanding the following derivations.

Once we have a set of n_b basis points ($n_b \leq n$), we construct the $<n_b \times n_b>$ matrix of pairwise overlaps between the basis-point Gaussian wave functions β_i (using Equation 4.8):

$$\mathbf{N} = \begin{bmatrix} \langle \beta_1 \,|\, \beta_1 \rangle & \langle \beta_1 \,|\, \beta_2 \rangle & \cdots & \langle \beta_1 \,|\, \beta_{n_b} \rangle \\ \langle \beta_2 \,|\, \beta_1 \rangle & \langle \beta_2 \,|\, \beta_2 \rangle & \cdots & \langle \beta_2 \,|\, \beta_{n_b} \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \beta_{n_b} \,|\, \beta_1 \rangle & \langle \beta_{n_b} \,|\, \beta_2 \rangle & \cdots & \langle \beta_{n_b} \,|\, \beta_{n_b} \rangle \end{bmatrix} \tag{5.1}$$

This basis-overlap matrix $\mathbf{N}$ is square, symmetric, real-valued, and positive definite, which gives it 2 important properties: its eigenvalues are all positive real values, and its eigenvectors are all real-valued and pairwise orthogonal.

The canonical eigendecomposition of $\mathbf{N}$ gives the relation:

$$\mathbf{N} = \mathbf{Q}\,\mathbf{\Lambda}\,\mathbf{Q}^\top \tag{5.2}$$

where the diagonal matrix $\mathbf{\Lambda}$ has the eigenvalues of $\mathbf{N}$ along its diagonal, and the orthogonal matrix $\mathbf{Q}$ has the L2-normalized eigenvectors of $\mathbf{N}$ in its columns.

We use the eigenvectors of $\mathbf{N}$ (that is, the columns of $\mathbf{Q}$) as the eigenstates of the quantum system. Each quantum eigenstate can be interpreted as a linear combination of the n_b basis-point Gaussians β_i.

Now, we want the quantum eigenstates to be orthonormal, meaning that the overlap of each eigenstate with itself should be 1 and the overlap between different eigenstates should be 0.

[4] Weinstein, M.; Horn, D. (2009). "Dynamic quantum clustering: a method for visual exploration of structures in data". Physical Review E. 80 (6): 066117.

For arbitrary linear combinations $\boldsymbol{\omega}_1$ and $\boldsymbol{\omega}_2$ of the basis-point Gaussians β_i, their overlap is:

$$\langle \omega_1 \,|\, \omega_2 \rangle \;=\; \boldsymbol{\omega}_1^{\top}\, \mathbf{N}\, \boldsymbol{\omega}_2 \tag{5.3}$$

Then, for the linear combinations represented by the L2-normalized eigenvectors $\mathbf{v_i}$ and $\mathbf{v_j}$ of $\mathbf{N}$ with eigenvalues λ_i and λ_j, their overlap is:

$$\langle v_i \,|\, v_j \rangle \;=\; \mathbf{v_i}^{\top} \mathbf{N}\, \mathbf{v_j} \;=\; \mathbf{v_i}^{\top} \lambda_j\, \mathbf{v_j} \;=\; \lambda_j\, \mathbf{v_i}^{\top} \mathbf{v_j} \;=\; \begin{cases} \lambda_j & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{5.4}$$

Thus, dividing each L2-normalized eigenvector $\mathbf{v_i}$ (in the columns of $\mathbf{Q}$) by $\sqrt{\lambda_i}$ gives us our orthonormal quantum eigenstates $\mathbf{e_i}$ (in the columns of $\mathbf{Q_e}$):

$$\mathbf{Q_e} \;=\; \mathbf{Q}\, \boldsymbol{\Lambda}^{-\frac{1}{2}} \tag{5.5}$$

We will henceforth refer to these orthonormal quantum eigenstates $\mathbf{e_i}$ collectively as the *eigenbasis*.

In practice, for numerical stability, eigenvectors with very small eigenvalues must be dropped (since we're dividing by the square root of the eigenvalue), which means that the number n_e of quantum eigenstates may be less than the number of basis points (so, $n_e \leq n_b$). As a simple example, some eigenstates will need to be dropped if a 'full' basis is used (i.e., all data points are in the basis) and the data set contains duplicate points. Thus, in practice, $\mathbf{Q_e}$ is actually an $<n_b \times n_e>$ matrix.

All wave functions can now be represented and manipulated as linear combinations of these eigenstates. In addition to the basis-point wave functions, other wave functions to be represented include:

- the initial Gaussian wave functions for non-basis points
- the initial Gaussian wave functions for any new, 'out-of-sample' points not present in the original data set
- all evolved wave functions (after $t = 0$)

Note that we have reduced the infinite-dimensional Hilbert space of wave functions to an n_e-dimensional space, and reconstructions of arbitrary wave functions in this space will in general be imperfect approximations (see Section 5.5).

Our 'eigenbasis' is not actually a basis, since a real basis would be able to represent any wave function perfectly (and would necessarily be infinite). But our limited, imperfect basis serves the same basic purpose as a real basis, and for all practical purposes we use it in the same way.

5.4 Representing Wave Functions in the Eigenbasis

To represent an arbitrary wave function ψ in the eigenbasis, we first build the vector of overlaps between ψ and each basis-point Gaussian β_i:

$$\boldsymbol{\psi}_b = \begin{bmatrix} \langle \beta_1 \,|\, \psi \rangle \\ \langle \beta_2 \,|\, \psi \rangle \\ \vdots \\ \langle \beta_{n_b} \,|\, \psi \rangle \end{bmatrix} \tag{5.6}$$

As mentioned at the beginning of Section 5.3, this vector can be loosely interpreted as the 'coordinates' of the wave function in the basis.

If ψ is a simple Gaussian wave function (Equation 4.4), then construction of the vector $\boldsymbol{\psi}_b$ is straightforward (using Equations 4.8 and 5.6).

Now we convert ψ from the basis ('b') to the eigenbasis ('e'), using the matrix $\mathbf{Q_e}$ (which contains the definitions of the eigenstates in terms of the basis-point Gaussians β_i):

$$\boldsymbol{\psi}_e \;=\; \mathbf{Q_e}^{\top}\, \boldsymbol{\psi}_b \tag{5.7}$$

$\boldsymbol{\psi}_e$ is now an $<n_e \times 1>$ vector, whose entries represent a weight for each quantum eigenstate. For any simple Gaussian wave function, this representation in the eigenbasis will have only real values. However, the weights will in general take on complex values during evolution.

The complex-valued weights in $\boldsymbol{\psi}_e$ are again interpretable as probability amplitudes, and their squared norms are interpretable as the probability of being in a given eigenstate. This means that generally we would like the L2 norm of $\boldsymbol{\psi}_e$ to be 1 (meaning, quantum mechanically, that ψ is guaranteed to be in some eigenstate when observed). However, the L2 norm of $\boldsymbol{\psi}_e$ will often be less than 1 (see Section 5.5).

5.5 Reconstruction of Wave Functions in the Eigenbasis

We have divided the infinite-dimensional Hilbert space of wave functions into two subspaces: the n_e-dimensional subspace defined by our eigenbasis, and the still infinite-dimensional 'complementary' subspace that is orthogonal to the first subspace.

Any wave function that is a linear combination of the basis-point Gaussians β_i can be perfectly reconstructed in the eigenbasis.

Any wave function ψ that is *not* a linear combination of the basis-point Gaussians β_i cannot be perfectly reconstructed in the eigenbasis, which is equivalent to saying that ψ will have a non-zero component ψ_c in the 'complementary' subspace. If ψ is normalized, this means that the norm of the component in the eigenbasis subspace, $\boldsymbol{\psi}_e$, is strictly less than 1.

In fact, the norm of $\boldsymbol{\psi}_e$ can be used as a measure of how well the wave function ψ is reconstructed in the eigenbasis. For example, consider a Gaussian wave function centered on a new data point far away from the entire original data set; its overlaps with all of the basis-point Gaussians will be very small, and so the norm if its $\boldsymbol{\psi}_e$ will be very small. This new point is simply not well represented by the basis, and so studying its behavior in DQM using this basis is not meaningful[5].

In practice, the process of choosing a threshold for which $\boldsymbol{\psi}_e$ norms are 'too small' is not precisely specified, may be context-dependent, and is a subject for further study.

6 Building the Quantum Operators

In this section, we develop the operators that will enable the quantum evolution of each data point. The operators will perform these actions for us in each iteration of the evolution:

- the evolution operator will 'evolve' the current state of each data point by producing a new, altered state (as represented in the eigenbasis).
- the position operators will operate on the new, evolved eigenbasis state to produce the new expected position of the data point in the data space.

In the implementation of DQM, these operators are actually matrices, which are applied to the eigenbasis-state vectors of the data points using simple matrix arithmetic (see Section 7).

6.1 Introduction to Quantum Operators

A *quantum operator* is a function that takes a wave function as input and produces a new wave function as output (where neither wave function is necessarily normalized). So, an operator $\hat{A}$ acts on a wave function ψ to produce a new wave function ψ':

$$\hat{A}\,|\,\psi\rangle \;=\; |\,\psi'\rangle \tag{6.1}$$

Given any set of basis wave functions $|\,u_i\rangle$, the wave function ψ can be represented in the basis as a vector with coordinates c_i:

$$\boldsymbol{\psi} \;=\; \begin{bmatrix} \langle u_1\,|\,\psi\rangle \\ \langle u_2\,|\,\psi\rangle \\ \vdots \end{bmatrix} \;=\; \begin{bmatrix} c_1 \\ c_2 \\ \vdots \end{bmatrix} \tag{6.2}$$

[5]Technically, one could choose a sufficiently large value of σ to cause the new point to be well represented. But this large value of σ would create a DQM landscape that, while it successfully 'captures' the new point, has washed away any interesting detail in the original data set.

(Note that Equation 6.2 is the more general form of Equation 5.6.)

For this basis, the operator $\hat{A}$ then has an associated *operator matrix* $\mathbf{A}$:

$$\mathbf{A} = \begin{bmatrix} \langle u_1 \,|\, \hat{A} \,|\, u_1 \rangle & \langle u_1 \,|\, \hat{A} \,|\, u_2 \rangle & \cdots \\ \langle u_2 \,|\, \hat{A} \,|\, u_1 \rangle & \langle u_2 \,|\, \hat{A} \,|\, u_2 \rangle & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \tag{6.3}$$

where each matrix entry A_{ij} is defined as:

$$A_{ij} = \langle u_i \,|\, \hat{A} \,|\, u_j \rangle = \langle u_i \,|\, u'_j \rangle = \int_{-\infty}^{\infty} u_i^*(\vec{x})\, u'_j(\vec{x})\, d\vec{x} \tag{6.4}$$

Then the operator matrix $\mathbf{A}$ produces the vector $\boldsymbol{\psi'}$ from the vector $\boldsymbol{\psi}$ by simple matrix arithmetic:

$$\mathbf{A}\,\boldsymbol{\psi} = \begin{bmatrix} A_{11} & A_{12} & \cdots \\ A_{21} & A_{22} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} c'_1 \\ c'_2 \\ \vdots \end{bmatrix} = \boldsymbol{\psi'} \tag{6.5}$$

6.2 Operator Functions

Often (not always), an operator $\hat{A}$ can be represented as an associated *operator function* $A(\vec{x})$, where:

$$\hat{A} \,|\, \psi \rangle = A(\vec{x})\, \psi(\vec{x}) = \psi'(\vec{x}) = |\, \psi' \rangle \tag{6.6}$$

That is, the value of $\psi'(\vec{x})$ at any given point in space is the value of $\psi(\vec{x})$ at that point times the value of $A(\vec{x})$ at that same point.

In this case, Equation 6.4 can be written as:

$$A_{ij} = \langle u_i \,|\, \hat{A} \,|\, u_j \rangle = \int_{-\infty}^{\infty} u_i^*(\vec{x})\, A(\vec{x})\, u_j(\vec{x})\, d\vec{x} \tag{6.7}$$

When dealing with specific operators in the following sections, we will replace an operator with its associated operator function whenever possible.

6.3 Quantum Operators in DQM

In the DQM context, it is much easier to build an operator matrix in the non-orthogonal basis of n_b Gaussian basis-point wave functions β_i and then convert to the orthonormal eigenbasis.

We construct an operator matrix $\mathbf{A_b}$ in the non-orthogonal basis ('b') like this:

$$\mathbf{A_b} = \begin{bmatrix} \langle \beta_1 \,|\, \hat{A} \,|\, \beta_1 \rangle & \langle \beta_1 \,|\, \hat{A} \,|\, \beta_2 \rangle & \cdots & \langle \beta_1 \,|\, \hat{A} \,|\, \beta_{n_b} \rangle \\ \langle \beta_2 \,|\, \hat{A} \,|\, \beta_1 \rangle & \langle \beta_2 \,|\, \hat{A} \,|\, \beta_2 \rangle & \cdots & \langle \beta_2 \,|\, \hat{A} \,|\, \beta_{n_b} \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \beta_{n_b} \,|\, \hat{A} \,|\, \beta_1 \rangle & \langle \beta_{n_b} \,|\, \hat{A} \,|\, \beta_2 \rangle & \cdots & \langle \beta_{n_b} \,|\, \hat{A} \,|\, \beta_{n_b} \rangle \end{bmatrix} \tag{6.8}$$

where each matrix entry $A_{b_{ij}}$ is defined as:

$$\begin{aligned} A_{b_{ij}} = \langle \beta_i \,|\, \hat{A} \,|\, \beta_j \rangle = \int_{-\infty}^{\infty} \beta_i^*(\vec{x})\, A(\vec{x})\, \beta_j(\vec{x})\, d\vec{x} = \\ {C_g}^2 \int_{-\infty}^{\infty} e^{-\frac{|\vec{x}-\vec{x}_i|^2}{2\sigma^2}} A(\vec{x})\, e^{-\frac{|\vec{x}-\vec{x}_j|^2}{2\sigma^2}}\, d\vec{x} \end{aligned} \tag{6.9}$$

Note that we're focusing here on the case of an operator $\hat{A}$ where the operator function $A(\vec{x})$ is known (see Section 6.2).

Using an intermediate result from Equation 4.8 and a variable substitution, we can rewrite the operator-matrix entries as:

$$\begin{aligned} A_{b_{ij}} = {C_g}^2 \int_{-\infty}^{\infty} e^{-\frac{|\vec{x}-\vec{x}_i|^2}{2\sigma^2}} A(\vec{x})\, e^{-\frac{|\vec{x}-\vec{x}_j|^2}{2\sigma^2}}\, d\vec{x} = \\ e^{-\frac{|\vec{x}_i-\vec{x}_j|^2}{4\sigma^2}} {C_g}^2 \int_{-\infty}^{\infty} e^{-\frac{|\vec{x}-\frac{\vec{x}_i+\vec{x}_j}{2}|^2}{2\sigma^2}} A(\vec{x})\, e^{-\frac{|\vec{x}-\frac{\vec{x}_i+\vec{x}_j}{2}|^2}{2\sigma^2}}\, d\vec{x} = \\ N_{ij}\, {C_g}^2 \int_{-\infty}^{\infty} e^{-\frac{|\vec{y}|^2}{2\sigma^2}} A(\vec{y} + \tfrac{\vec{x}_i+\vec{x}_j}{2})\, e^{-\frac{|\vec{y}|^2}{2\sigma^2}}\, d\vec{y} \end{aligned} \tag{6.10}$$

In practice, we simplify further by using the leading term of a Taylor series expansion of $A(\vec{x})$ around the basis-basis midpoint $\frac{\vec{x}_i+\vec{x}_j}{2}$:

$$A_{b_{ij}} = N_{ij}\, A(\tfrac{\vec{x}_i+\vec{x}_j}{2}) \left({C_g}^2 \int_{-\infty}^{\infty} e^{-\frac{|\vec{y}|^2}{2\sigma^2}} e^{-\frac{|\vec{y}|^2}{2\sigma^2}}\, d\vec{y} \right) = N_{ij}\, A(\tfrac{\vec{x}_i+\vec{x}_j}{2}) \tag{6.11}$$

It is now easy to build the operator matrix $\mathbf{A_b}$ by evaluating $A(\vec{x})$ at each basis-basis midpoint and multiplying elementwise by the basis-overlap matrix $\mathbf{N}$.

Using additional terms from the Taylor series would improve accuracy but would also make the implementation of DQM both slower and more complex. Assessing just how much additional terms would alter final DQM results is a subject for further study.

Finally, we convert the operator matrix from the non-orthogonal basis ('b') to the eigenbasis ('e'), using the matrix $\mathbf{Q_e}$ (which, remember, contains the definitions of the orthonormal eigenstates $\mathbf{e_i}$ in terms of the basis-point Gaussians β_i):

$$\mathbf{A}_e = \mathbf{Q}_e^\top \mathbf{A}_b \mathbf{Q}_e \tag{6.12}$$

Note that $\mathbf{A}_b$ is an $<n_b \times n_b>$ matrix and $\mathbf{A}_e$ is an $<n_e \times n_e>$ matrix.

6.4 Building the Position Operators

Our ultimate goal is to build a trajectory of positions through the data space for each point. At each time step in the evolution, then, we need to find the expected position for each evolved wave function $\psi_j(\vec{x}, t)$. Remembering that the squared norm of the wave function at a point in space is the probability density ρ, we find the expected position of $\psi_j(\vec{x}, t)$ by calculating the probability-density-weighted sum of positions over all of space:

$$\begin{gathered}\langle X \rangle = \int_{-\infty}^{\infty} \vec{x}\, \rho(\vec{x}, t)\, d\vec{x} = \int_{-\infty}^{\infty} \psi_j(\vec{x}, t)^*\, \vec{x}\, \psi_j(\vec{x}, t)\, d\vec{x} = \\ \langle \psi_j(\vec{x}, t) \,|\, \hat{X} \,|\, \psi_j(\vec{x}, t) \rangle\end{gathered} \tag{6.13}$$

where the position operator $\hat{X}$ simply multiplies a wave function by its position at each point in space:

$$\hat{X} \,|\, \psi \rangle = X(\vec{x})\, \psi(\vec{x}) = \vec{x}\, \psi(\vec{x}) \tag{6.14}$$

Since $\vec{x}$ is a d-dimensional vector, in practice we find the expected position of a wave function independently in each dimension. In the kth dimension, then, we use the operator $\hat{K}$, which multiplies a wave function at every point in space by the kth position coordinate at the same point in space:

$$\hat{K} \,|\, \psi \rangle = K(\vec{x})\, \psi(\vec{x}) = x_k\, \psi(\vec{x}) \tag{6.15}$$

and the expected position for the wave function in the kth dimension is:

$$\begin{gathered}\langle K \rangle = \langle \psi(\vec{x}, t) \,|\, \hat{K} \,|\, \psi(\vec{x}, t) \rangle = \\ \int_{-\infty}^{\infty} \psi(\vec{x}, t)^*\, x_k\, \psi(\vec{x}, t)\, d\vec{x} = \int_{-\infty}^{\infty} x_k\, \rho(\vec{x}, t)\, d\vec{x}\end{gathered} \tag{6.16}$$

where the integral is still over all of space, in all d dimensions.

We now build $\mathbf{K}_e$, the eigenbasis kth-dimension position-operator matrix, following the framework of Section 6.3.

The operator function for $\hat{K}$ is simply $K(\vec{x}) = x_k$, and so building the non-orthogonal operator matrix $\mathbf{K}_b$ is straightforward, using Equation 6.11:

$$\mathbf{K}_{b_{ij}} = \mathbf{N}_{ij} \left(\frac{x_{k_i} + x_{k_j}}{2}\right) \tag{6.17}$$

where x_{k_i} and x_{k_j} are the coordinates in the kth dimension of the centers of the basis-point Gaussians β_i and β_j, respectively.

We then convert to the eigenbasis using Equation 6.12:

$$\mathbf{K}_e = \mathbf{Q}_e^\top \mathbf{K}_b \mathbf{Q}_e \tag{6.18}$$

Finally, the matrix calculation for the kth coordinate of the expected position of ψ is:

$$\langle K \rangle = \langle \psi(\vec{x}, t) \,|\, \hat{K} \,|\, \psi(\vec{x}, t) \rangle = \boldsymbol{\psi}_e^\dagger \, \mathbf{K}_e \, \boldsymbol{\psi}_e \tag{6.19}$$

where $\boldsymbol{\psi}_e^\dagger$ is the complex-conjugate transpose of the vector $\boldsymbol{\psi}_e$. (Remember that the entries of $\boldsymbol{\psi}_e$ may in general have complex values.)

We now have what we need: once we have built the $\mathbf{K}_e$ matrices for each of the d dimensions, repeated application of Equation 6.19 during evolution will give us our sequence of expected positions (see Section 7.4).

Ultimately, this framework can only produce expected positions whose coordinates are linear combinations of the coordinates of the original basis-point positions. This is the sense in which the size of the basis represents a resolution for the landscape. In the most trivial examples: for 2 basis points, all representable positions lie on a line connecting the 2 basis points; for 3 basis points, all representable positions lie in the triangle defined by the 3 basis points. For, say, 1,000 basis points, a great deal more nuance is available for describing possible positions (but only in the region of the space where the basis points lie).

6.5 Building the Quantum Potential

We now build a quantum potential V that represents the landscape defined by our data set. Regions of higher data density will correspond to lower energy levels in the potential, and individual data points will move downhill in this potential during the DQM evolution.

To create the potential, we assume that the overall wave function for the data set (Equation 4.6) represents a stable state of the system. This means that neither the potential V nor the overall wave function ψ_f depends on time, and together they satisfy the time-independent Schrödinger wave equation:

$$\hat{H} \,|\, \psi_f \rangle = E \,|\, \psi_f \rangle \tag{6.20}$$

where E is the total scalar energy of the system and $\hat{H}$ is the Hamiltonian operator, which corresponds to the total energy of the system and includes both a kinetic-energy term and a potential-energy term:

$$\hat{H} = -\frac{\hbar^2}{2m}\nabla^2 + \hat{V} \tag{6.21}$$

The kinetic-energy term of the Hamiltonian uses the d-dimensional Laplacian operator ∇^2, which is the sum of second-order spatial derivatives:

$$\nabla^2 \;=\; \frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2} + \ldots + \frac{\partial^2}{\partial x_d^2} \tag{6.22}$$

The 'kinetic-energy' term here involves spatial derivatives of the wave function, not temporal derivatives, which is strange from the standpoint of classical physics. Explaining this important aspect of quantum mechanics is beyond the scope of this paper, but much can be read about it elsewhere.

The potential-energy term of the Hamiltonian is simpler: the potential operator $\hat{V}$ simply multiplies a wave function by the value of the potential at each point in space. Thus, the associated operator function is simply the potential function $V(\vec{x})$.

6.5.1 Deriving The Potential Function

Here we derive a usable formula for calculating the potential function $V(\vec{x})$ at any given point in space.

Changing the total energy of the system has no effect on the dynamics of the DQM evolution. (Changing the total energy moves the entire potential landscape 'up' or 'down' without changing its shape in any way. But the shape of the landscape is all we care about.) So, for mathematical simplicity, we simply set $E = 0$.

Combining Equations 6.20 and 6.21, and with $E = 0$, we then have:

$$-\frac{\hbar^2}{2m}\nabla^2\psi_f(\vec{x}) + V(\vec{x})\,\psi_f(\vec{x}) \;=\; 0 \tag{6.23}$$

Note that we have replaced the potential operator with its associated potential function, but ∇^2 is still an operator.

Solving for $V(\vec{x})$, Equation 6.23 gives us:

$$V(\vec{x}) \;=\; \frac{\hbar^2\,\nabla^2\psi_f(\vec{x})}{2\,m\,\psi_f(\vec{x})} \tag{6.24}$$

Changing the overall scale of the potential also does not change the DQM dynamics (for similar reasons as with the total energy, above – changing the 'size' or scale of the landscape does not change its shape). So, again for mathematical simplicity, we set $\hbar^2/\,2m = 1$, which gives us:

$$V(\vec{x}) \;=\; \frac{\nabla^2\psi_f(\vec{x})}{\psi_f(\vec{x})} \tag{6.25}$$

Note that if $\psi_f(\vec{x})$ is multiplied by any non-zero scalar, that scalar appears in both the numerator and denominator of Equation 6.25 and so cancels out.

This is why we don't need to worry about normalizing the overall wave function of the data set.

For the individual Gaussian wave function ψ_j (for the jth data point), the Laplacian of the wave function is:

$$\nabla^2\psi_j(\vec{x}) = \frac{1}{\sigma^4}|\vec{x}-\vec{x}_j|^2\, C_g\, e^{-\frac{|\vec{x}-\vec{x}_j|^2}{2\sigma^2}} - \frac{d}{\sigma^2}\, C_g\, e^{-\frac{|\vec{x}-\vec{x}_j|^2}{2\sigma^2}} = \frac{1}{\sigma^4}|\vec{x}-\vec{x}_j|^2\, \psi_j(\vec{x}) - \frac{d}{\sigma^2}\, \psi_j(\vec{x}) \tag{6.26}$$

The overall wave function ψ_f is the sum of all the individual wave functions ψ_j, and the Laplacian operator is distributive over addition. So, combining Equations 6.25 and 6.26, the potential function is now:

$$V(\vec{x}) = \frac{\frac{1}{\sigma^4}\sum_j |\vec{x}-\vec{x}_j|^2\, \psi_j(\vec{x}) - \frac{d}{\sigma^2}\, \sum_j \psi_j(\vec{x})}{\sum_j \psi_j(\vec{x})} = \frac{1}{\sigma^4}\frac{\sum_j |\vec{x}-\vec{x}_j|^2\, \psi_j(\vec{x})}{\sum_j \psi_j(\vec{x})} - \frac{d}{\sigma^2} \tag{6.27}$$

Again, neither the absolute 'height' nor the absolute scale of the potential landscape affects the DQM dynamics, and so we can simplify further by dropping both the constant term and the coefficient[6]:

$$V(\vec{x}) = \frac{\sum_j |\vec{x}-\vec{x}_j|^2\, \psi_j(\vec{x})}{\sum_j \psi_j(\vec{x})} = \frac{\sum_j |\vec{x}-\vec{x}_j|^2\, C_g\, e^{-\frac{|\vec{x}-\vec{x}_j|^2}{2\sigma^2}}}{\sum_j C_g\, e^{-\frac{|\vec{x}-\vec{x}_j|^2}{2\sigma^2}}} = \frac{\sum_j |\vec{x}-\vec{x}_j|^2\, e^{-\frac{|\vec{x}-\vec{x}_j|^2}{2\sigma^2}}}{\sum_j e^{-\frac{|\vec{x}-\vec{x}_j|^2}{2\sigma^2}}} \tag{6.28}$$

In practice, we use the final form of Equation 6.28 to evaluate the potential $V(\vec{x})$ at any particular point in space.

Note that the sums in Equation 6.28 use *all* data points, not just the basis points. Since the potential landscape itself does not change as the DQM evolution proceeds, we only need to calculate values for the potential once when building the potential-operator matrix (see Section 6.5.2), and so using all data points is computationally feasible (even for millions of points).

[6]It may seem problematic to drop coefficients that include factors of σ. However, when building a given DQM landscape, σ will always have a single fixed value. This value of σ will be critically important in relation to the distances between the various data points in the space, but the presence of σ in overall height and scale parameters for the potential are not part of this dynamic.

6.5.2 Building the Potential-Operator Matrix

We're going to build an operator matrix **H** for the Hamiltonian (see Section 6.6.1), and so we'll need an operator matrix **V** for the potential.

We build the non-orthogonal operator matrix $\mathbf{V}_b$ just as we did for the position operators, using Equations 6.11 and 6.28:

$$\mathbf{V}_{b_{ij}} \;=\; \mathbf{N}_{ij}\, V(\frac{\vec{x}_i + \vec{x}_j}{2}) \tag{6.29}$$

where, again, evaluation of the potential function $V(\vec{x})$ at each basis-basis midpoint includes contributions from all data points, not just basis points.

We don't need to convert to the eigenbasis yet; $\mathbf{V}_b$ will be incorporated into the Hamiltonian before conversion (see Section 6.6.1).

6.6 Building the Evolution Operator

Now that we have our potential landscape, the last piece of the puzzle is the process of evolving each individual data point's wave function $\psi_j(\vec{x}, t)$ within that landscape. ($\psi_j(\vec{x}, 0)$ is the initial Gaussian for the jth data point, Equation 4.4.)

We evolve the individual wave functions using the time-dependent Schrödinger wave equation:

$$i\,\frac{\partial}{\partial t}\,|\,\psi_j\rangle \;=\; \hat{H}\,|\,\psi_j\rangle \tag{6.30}$$

We omit a factor of $\hbar$ from the left side of the equation, which would ultimately be absorbed into the choice of time step (see below).

Importantly, we hold the initial potential landscape fixed; it does not change as the data points move through time. This means that the Hamiltonian operator itself does not depend on time, and so Equation 6.30 has a simple solution:

$$|\,\psi_j(\vec{x}, t_0 + t)\rangle \;=\; e^{-i\hat{H}t}\,|\,\psi_j(\vec{x}, t_0)\rangle \tag{6.31}$$

We will refer to the operator $\hat{E} \;=\; e^{-i\hat{H}t}$ as the 'evolution' operator. Note that the value of t here is actually a time step, not an absolute time; applying the operator $\hat{E}$ evolves a wave function from any time t_0 to the time $t_0 + t$.

Choosing a size for the time step t is beyond the scope of this paper. In practice, a reasonable default value for t can be chosen that almost never needs to be changed. (See the *DQM User Guide* for more on this subject.)

6.6.1 Building the Hamiltonian-Operator Matrix

In order to build the operator matrix $\mathbf{E}_e$ for the evolution operator $\hat{E} = e^{-i\hat{H}t}$, we first need to build the operator matrix $\mathbf{H}_e$ for the Hamiltonian operator $\hat{H}$ (Equation 6.21). We proceed by building $\mathbf{H}_b$ and then converting $\mathbf{H}_b$ to $\mathbf{H}_e$ at the end. We've already constructed $\mathbf{V}_b$ (Equation 6.29), but we still need to build the operator matrix $\mathbf{L}_b$ for the Laplacian operator ∇^2 (Equation 6.22).

From Equation 6.26, we have:

$$\nabla^2 \psi_j(\vec{x}) = \left(\frac{|\vec{x} - \vec{x}_j|^2}{\sigma^4} - \frac{d}{\sigma^2} \right) \psi_j(\vec{x}) \tag{6.32}$$

We can drop the coefficient of $1/\sigma^4$, which would ultimately be absorbed into the choice of mass (see below). We can also drop the constant term of $-d/\sigma^2$, because the absolute total energy of an individual point's wave function does not affect its evolution[7]. So, we now have:

$$\nabla^2 \psi_j(\vec{x}) = |\vec{x} - \vec{x}_j|^2 \, \psi_j(\vec{x}) \tag{6.33}$$

and so for the Laplacian operator ∇^2 we now know the associated operator function $L(\vec{x})$:

$$L(\vec{x}) = |\vec{x} - \vec{x}_j|^2 \tag{6.34}$$

Note that this function has a specific term $\vec{x}_j$ which comes from the initial Gaussian wave function ψ_j to which the Laplacian operator was applied.

We are now ready to construct the non-orthogonal Laplacian operator matrix $\mathbf{L}_b$ using Equation 6.11:

$$L_{b_{ij}} = N_{ij} \, L(\tfrac{\vec{x}_i + \vec{x}_j}{2}) = N_{ij} \, |\tfrac{\vec{x}_i + \vec{x}_j}{2} - \vec{x}_k|^2 \tag{6.35}$$

Here, i and j are now both basis points, and k is the wave function to which the Laplacian operator was applied. In fact, k will always be either i or j (see Equation 6.8), and either way the result will be the same:

$$L_{b_{ij}} = N_{ij} \, \frac{|\vec{x}_i - \vec{x}_j|^2}{4} \tag{6.36}$$

[7] In Section 6.5.1, we employed the classical analogy of a physical landscape to see intuitively that the absolute 'height' of the quantum potential has no relationship to its shape. Here, we don't have a similarly convenient classical analogy to understand why the absolute total energy in an individual point's wave function does not affect how it evolves within the potential. Explaining this phenomenon is beyond the scope of this paper. (However, the result can be verified empirically by adding an arbitrary constant to the final Hamiltonian matrix and seeing that it does not change the final DQM results.)

Once again, we drop the factor of 1/4, which would ultimately be absorbed into the choice of mass (see below), and so we have:

$$L_{b_{ij}} = N_{ij} \, |\vec{x}_i - \vec{x}_j|^2 \tag{6.37}$$

We are now ready to construct the non-orthogonal Hamiltonian operator matrix $\mathbf{H}_b$, using Equations 6.21, 6.29, and 6.37:

$$\mathbf{H}_b = -\frac{\hbar^2}{2m} \mathbf{L}_b + \mathbf{V}_b \tag{6.38}$$

Yet again, we absorb the factor of $\hbar^2/2$ into the final choice of mass (see below), and so we have:

$$\mathbf{H}_b = -\frac{1}{m} \mathbf{L}_b + \mathbf{V}_b \tag{6.39}$$

Conceptually, m here is the mass of each individual data point as it evolves in the quantum-potential landscape. DQM has only a single value of m, which is applied to all points. The value of m controls the 'transparency' of the landscape for each point; smaller m makes tunneling through potential barriers easier, and larger m makes it harder.

Choosing the value of m is beyond the scope of this paper. In practice, a reasonable default value of m can be chosen, which needs to be changed far less often than σ. (See the *DQM User Guide* for more on this subject.)

We have now seen all three of DQM's tunable parameters: the Gaussian width σ, the time step t, and the point mass m. In practice, the DQM user will typically change σ frequently, m more rarely, and t very rarely (if ever).

We now make one last adjustment to the Hamiltonian, adding a factor of $1/\sigma^2$:

$$\mathbf{H}_b = \left(-\frac{1}{m} \mathbf{L}_b + \mathbf{V}_b \right) \sigma^{-2} \tag{6.40}$$

We do this because both terms of the Hamiltonian have scaling factors of $|\vec{x}|^2$ (in Equation 6.37 for the kinetic term and Equation 6.28 for the potential term); adding this factor of $1/\sigma^2$ thus makes the Hamiltonian scale-invariant. (We can use σ as a proxy for the overall scale of the data because useful/interesting values of σ will always be relative to that scale[8].) Without this scale-invariance, a reasonable value for the time step t (see Section 6.6) would depend on the overall scale of the data.

[8] Using σ here to create scale-invariance is convenient, but it couples the values of σ and t in a slightly inelegant way. It might be better to use some other scale metric, like the mean pairwise distance between points in the data set. (The cost of this approach would be a bit of added complexity in the implementation of the algorithm).

Finally, we are ready to convert the Hamiltonian operator matrix to the eigenbasis using Equation 6.12:

$$\mathbf{H}_e \;=\; \mathbf{Q}_e^{\top}\,\mathbf{H}_b\,\mathbf{Q}_e \tag{6.41}$$

6.6.2 Building the Evolution-Operator Matrix

Equation 6.31 gives us our definition of the evolution operator:

$$\hat{E} \;=\; e^{-i\hat{H}t} \tag{6.42}$$

We've built the Hamiltonian eigenbasis operator matrix $\mathbf{H}_e$ (Section 6.6.1), which means we're ready to build the evolution eigenbasis operator matrix $\mathbf{E}_e$.

How do we handle an operator in an exponent? Algebraically, this issue is often best handled using a Taylor series expansion of the exponential function. Here, though, working with the operator matrices, we take a (mathematically valid) shortcut: we diagonalize the operator matrix $\mathbf{H}_e$, exponentiate each element (i.e., each eigenvalue of $\mathbf{H}_e$) along the diagonal, and then undiagonalize again.

First, we diagonalize $\mathbf{H}_e$ using the canonical eigendecomposition (as in Equation 5.2):

$$\mathbf{H}_e \;=\; \mathbf{Q}_H\,\mathbf{\Lambda}_H\,\mathbf{Q}_H^{\top} \tag{6.43}$$

where $\mathbf{\Lambda}_H$ is a diagonal matrix with the eigenvalues of $\mathbf{H}_e$, and $\mathbf{Q}_H$ is an orthogonal matrix with the eigenvectors of $\mathbf{H}_e$ in its columns.

We need to solve for the diagonal eigenvalue matrix $\mathbf{\Lambda}_H$ (in practice, any eigenvalue solver will do this for us):

$$\mathbf{\Lambda}_H \;=\; \mathbf{Q}_H^{\top}\,\mathbf{H}_e\,\mathbf{Q}_H \tag{6.44}$$

For increased numerical stability, at this point we subtract the first eigenvalue from all the eigenvalues, making the ground-state energy of the system zero. (Quantum-mechanically, the eigenvalues of the Hamiltonian are, in fact, the possible energy levels for the individual point being represented.) This step does not affect the DQM evolution because the absolute total energy of the individual point is irrelevant. (See the footnoted comment in Section 6.6.1.)

Now, following Equation 6.42, we exponentiate the eigenvalues along the diagonal:

$$\Lambda_{E_{ii}} \;=\; e^{-i\Lambda_{H_{ii}}t} \tag{6.45}$$

And finally we undiagonalize again to produce the final form of the evolution eigenbasis operator matrix $\mathbf{E}_e$:

$$\mathbf{E}_e \;=\; \mathbf{Q}_H\,\mathbf{\Lambda}_E\,\mathbf{Q}_H^{\top} \tag{6.46}$$

7 Evolving the Data-Point Wave Functions

We're done building the DQM quantum operators, and now we'll use them to create the DQM evolution of each individual data point.

Each iteration of the evolution has several steps, described in the subsections below. The descriptions below all deal with some arbitrary data point. (The evolution process as described applies equally to all data points: basis points, non-basis points, and new data points that were not in the original data set.)

During the evolutionary process, all data points are entirely independent of each other. Thus, for computational efficiency, points can be evolved in parallel batches (on multiple machines, if desired), and new points can be evolved one at a time if needed.

Evolution can continue until all points have stopped moving, or can be cut off earlier if behavior of interest has already been identified. (See Section 7.1.1 below to understand why all points eventually do, in fact, stop moving.)

7.1 Representing Current Positions in the Eigenbasis

At the beginning of each iteration of the evolution, a given data point has some position in the data space: either its initial position, or its new expected position at the end of the previous iteration.

Given this position in space, we find the data point's current representation in the non-orthogonal basis using Equations 4.8 and 5.6. We then convert to the eigenbasis using Equation 5.7. (Using these equations assumes that the data point's current wave function is a simple Gaussian with width σ. See Section 7.1.1 below for more on this point.)

We now have a vector $\boldsymbol{\psi}_e$ representing the data point's current state in the eigenbasis.

7.1.1 Reconstructing Evolved Points as Gaussians

In Section 7.3 below, we will use the evolution operator to take an eigenbasis state vector $\boldsymbol{\psi}_e$ and create a new, 'evolved' eigenbasis state vector $\boldsymbol{\psi}'_e$. So, at the end of each iteration, we have an eigenbasis state vector describing the current (new) state of each data point. However, in Section 7.1 above we just said that we create an eigenbasis state vector from the point's position at the beginning of *every* iteration, not just the first one. After the first iteration, why don't we just use the evolved eigenbasis state from the end of the previous iteration? Why do we construct a new one each time?

The answer is that building a new state vector from the point's current position, as we do in Section 7.1 above, implicitly reconstitutes the point's wave function as a simple multidimensional Gaussian of width σ, centered on its current position. (Equation 4.8 in particular assumes the Gaussian nature of

the wave function.) Beginning each iteration with a simple Gaussian wave function solves two problems.

Problem 1: Packet Spreading

The phenomenon of *packet spreading* in quantum mechanics is beyond the scope of this paper. Suffice it to say here that, if we apply the evolution operator repeatedly to a data point's initial Gaussian wave function, the evolved wave function will look less and less like a Gaussian as it evolves, and the Gaussian-based framework we've built for describing and manipulating it will be less and less accurate.

We avoid this problem by beginning each iteration of the evolution with a simple Gaussian wave function centered on the point's current location.

Problem 2: Harmonic Oscillation

A simple Gaussian wave function is at rest; it has no kinetic component. (Explaining why this is true is, again, beyond the scope of this paper.) However, the evolved non-Gaussian wave function at the end of each iteration does have a kinetic component (directly analogous to the situation in classical physics where a ball picks up speed as it rolls down a hill). By reconstructing the wave function as a simple Gaussian, then, we throw away any kinetic component of the evolved wave function; in other words, the data point begins each iteration at rest.

Importantly, this is why, in DQM evolution, all data points will eventually come to a stop[9]; they would otherwise oscillate endlessly around their final equilibrium positions.

7.2 Renormalizing the State Vector

As discussed in Section 5.5, $\boldsymbol{\psi}_e$ will generally be an imperfect representation of the data point's Gaussian wave function, and we will usually have $|\boldsymbol{\psi}_e| < 1$.

Thus, we need to normalize $\boldsymbol{\psi}_e$, simply by dividing by its scalar L2 norm $|\boldsymbol{\psi}_e|$.

Conceptually, this returns the wave function to a normalized state where the probabilities are exactly 1 of finding the data point a) in some particular eigenstate of the quantum system, and b) at some particular position in the data space.

Without this renormalization step, data points would 'decay' toward the origin of the data space as they evolved.

[9] In practice, DQM has a 'stopping threshold'; points approach their final equilibrium positions asypmtotically slowly, and DQM considers that they have stopped when their 'speed' of motion from one iteration to the next has dropped below the threshold. (This approximation means that the DQM evolution will miss the occasional very late tunneling event, which is a small price to pay for a great increase in computational efficiency.)

7.3 Applying the Evolution Operator

We've done all of the hard work, and now applying the evolution operator is extremely simple. Following Equation 6.5, we apply the evolution eigenbasis operator matrix $\mathbf{E}_e$ to create the data point's evolved state vector $\boldsymbol{\psi}'_e$ from its current state vector $\boldsymbol{\psi}_e$:

$$\boldsymbol{\psi}'_e \;=\; \mathbf{E}_e \, \boldsymbol{\psi}_e \tag{7.1}$$

The 'e' subscripts are a reminder that all representations here are in the eigenbasis.

7.4 Applying the Position Operators

Now that we have the evolved state vector $\boldsymbol{\psi}'_e$, we can apply the position operators to find the data point's new expected position. Equation 6.19 gives us the data point's new expected position for the kth dimension in the data space:

$$\langle K \rangle \;=\; \langle \psi'(\vec{x},t) \,|\, \hat{K} \,|\, \psi'(\vec{x},t) \rangle \;=\; \boldsymbol{\psi}'^{\dagger}_e \, \mathbf{K}_e \, \boldsymbol{\psi}'_e \tag{7.2}$$

where $\boldsymbol{\psi}'^{\dagger}_e$ is the complex-conjugate transpose of the vector $\boldsymbol{\psi}'_e$. (We start each iteration with a simple Gaussian wave function, but the entries of $\boldsymbol{\psi}'_e$ will in general have complex values after evolution.)

We now have the data point's new 'current' position in the data space, and this iteration of the evolution is complete.

8 Conclusion

The final output of the DQM evolution is a sequence of positions for every evolved data point (which, again, may be either some or all of the original data points, a new set of data points, or any mixture thereof). The actual result is a 3-dimensional $<n \times d \times f>$ tensor matrix, where n is the number of evolved data points, d is the number of dimensions of the data space, and f is the number of 'frames' (iterations) of the evolution.

As described in Section 2, these results provide a rich source of information about the high-dimensional structure of the data set. There is much to say about using and interpreting DQM results, for which see the *DQM User Guide*.

Appendices

A Ehrenfest's Theorem

Ehrenfest's Theorem is a well known result in theoretical quantum mechanics. It says that the expectation values of certain quantum properties obey the laws of classical physics. More specifically, it gives us this relation:

$$-\langle\, \vec{\nabla} V(\vec{x}) \,\rangle \;=\; m\, \frac{d^2}{dt^2}\, \langle\, \vec{x}(t) \,\rangle \tag{A.1}$$

which is directly analogous to Newton's Second Law in classical physics:

$$\vec{F} \;=\; m\, \vec{a} \tag{A.2}$$

where $\vec{F}$ is the force acting on a particle, m is the mass of the particle, and $\vec{a}$ is the acceleration of the particle.

The analogy is completed by the well known fact in classical physics that the force acting on a particle in a potential is the negative gradient $(-\vec{\nabla})$ of the potential.

Importantly, the analogy only holds for the *expectation values* of the quantum properties. The fact that our quantum 'points' move downhill in the quantum potential *in expectation* is why the entire 'landscape' analogy applies to the process of DQM evolution. The fact that they do so *only* in expectation is why we have the phenomena of non-local gradient descent and quantum tunneling, which provide an elegant solution to the problem of non-convex gradient descent.

B Kernel Methods

The entire mathematical framework of DQM comes from the world of quantum mechanics. However, there are strong similarities between certain aspects of the quantum framework used in DQM and the technique of *kernel methods* used in machine learning. (Support-vector machines are perhaps the most widely known example of use of kernel methods in machine learning).

Further elucidation of this parallel might yield valuable insights, including how aspects of the DQM framework might be useful in other areas of machine learning.