A quantum calculus formulation of dynamic programming and ordered derivatives

John E. Seiffertt IV  
*Missouri University of Science and Technology, jes0b4@mst.edu*

Donald C. Wunsch  
*Missouri University of Science and Technology, dwunsch@mst.edu*

Follow this and additional works at: [http://scholarsmine.mst.edu/faculty_work](http://scholarsmine.mst.edu/faculty_work)  
Part of the [Electrical and Computer Engineering Commons](http://scholarsmine.mst.edu/faculty_work)

Recommended Citation  
[http://scholarsmine.mst.edu/faculty_work/949](http://scholarsmine.mst.edu/faculty_work/949)

This Article - Conference proceedings is brought to you for free and open access by Scholars' Mine. It has been accepted for inclusion in Faculty Research & Creative Works by an authorized administrator of Scholars' Mine. For more information, please contact weaverjr@mst.edu.
A Quantum Calculus Formulation of Dynamic Programming and Ordered Derivatives

John Seiffertt, Student Member, IEEE and Donald C. Wunsch II, Fellow, IEEE

Abstract—Much recent research activity has focused on the theory and application of quantum calculus. This branch of mathematics continues to find new and useful applications and there is much promise left for investigation into this field. We present a formulation of dynamic programming grounded in the quantum calculus. Our results include the standard dynamic programming induction algorithm which can be interpreted as the Hamilton-Jacobi-Bellman equation in the quantum calculus. Furthermore, we show that approximate dynamic programming in quantum calculus is tenable by laying the groundwork for the backpropagation algorithm common in neural network training. In particular, we prove that the chain rule for ordered derivatives, fundamental to backpropagation, is valid in quantum calculus. In doing this we have connected two major fields of research.

Index Terms—dynamic programming, quantum calculus, time scales, backpropagation, dynamic equations

I. INTRODUCTION

Quantum calculus is the modern name for the investigation of the calculus without limits which began with Euler and currently enjoys ties to abstract algebra and has found application in the quantum mechanics literature. The book by Kac and Cheung [20] covers many of the fundamental aspects of the quantum calculus. As this field becomes more widely researched, an increasing number of application areas are being discovered. For example, a recent study of financial derivative pricing realized a quantum calculus analog of the Black-Scholes equation [22]. Additionally, it has been shown that quantum calculus is a subfield of the more general mathematical field of time scales calculus. Time scales provide a unified framework for studying dynamic equations on both discrete and continuous domains. The texts by Bohner and Peterson ([11],[12]) collect much of the core theory in the calculus of time scales. Applications of this mathematics include population biology [7], geometric analysis [10], real time communications networks [18], intelligent robotic control [17], adaptive sampling [16], approximation theory [28], macroeconomics [1], and financial engineering [27], among others. Many of these are ideal areas for Approximate Dynamic Programming (ADP) ([23],[29]).

Dynamic programming itself has been extensively applied in computational economics ([21],[30],[32]) and asset price modeling [13], among a host of engineering topics with which we assume the reader is familiar. Thus, with the recent work on quantum calculus applications to finance, it is worthwhile to extend the investigation to other computational methods which have enjoyed success in economics and finance. Dynamic programming concerns itself with allocation of resources under uncertainty ([2],[3]) and we believe that time scales calculus, and quantum calculus in particular, may be the appropriate mathematical framework in which to cast an important class of decision problems.

This paper presupposes familiarity with the core ideas of dynamic programming. Quantum calculus, however, is introduced and reviewed in section II and the basic structure of a dynamic programming problem as well as derivation of the dynamic programming algorithm are presented in section III. Section IV provides the foundation for backpropagation on q-time scales by proving the chain rule for ordered derivatives originally introduced for the continuous case by Werbos. Section V concludes with notes on further research that can help leverage this newly developing area of mathematics into applications of engineering interest.

II. OVERVIEW OF THE QUANTUM CALCULUS

A time scale \( \mathbb{T} \) is any nonempty closed subset of the real line \( \mathbb{R} \). Examples include the integers \( \mathbb{Z} \), the scaled integers \( h\mathbb{Z} \) (where \( h > 0 \) is a scaling factor), as well as more mathematically adventurous entities such as the Cantor set. In studying quantum calculus we are concerned with a specific time scale, called the \( q \)-time scale, defined as follows:

\[
\mathbb{T} = q^\mathbb{Z} = \{q^k : k \in \mathbb{Z}\}
\]

such that \( q > 1 \). Dynamic equations in the quantum calculus, then, have domain \( q^\mathbb{Z} \). It is worth noting that the quantum calculus converges to the classical continuous calculus in the limit as \( q \) approaches 1 from above.

Important in the study of any time scale are three characteristic functions. Let \( \mathbb{T} \) be a time scale. The forward jump operator \( \sigma \) is defined as \( \sigma(t) = \inf \{x \in \mathbb{T} : x > t\} \). The
The purpose of this function is to step through to the successor element of the time scale, in the sense given in its definition. For the time scale $\mathbb{T} = \mathbb{Z}$, $\sigma(t) = t + 1$. The $q$-time scale analog of the forward jump operator is given by $\sigma(t) = qt$ as the $q$-time scale is said to be isolated. In a similar fashion, the backward jump operator $\rho$ is defined as $\rho(t) = \text{sup}\{x \in \mathbb{T} : x < t\}$. Central to many formulas of the time scales calculus, the graininess function $\mu$ is given by $\mu(t) = \sigma(t) - t$. Note that when $\mathbb{T} = \mathbb{Z}$, $\mu(t) = t$ and when $\mathbb{T} = \mathbb{R}$, $\mu(t) = 0$. The graininess of the $q$-time scale can be shown to be $\mu(t) = q^t(q - 1)$ via application of the definition of the forward jump operator $\sigma$ and some algebra.

To discuss calculus on the $q$-time scale we need to define a derivative. The $q$-differential of a function $f$ is given by

$$d_q f(x) = f(qx) - f(x)$$

and the $q$-derivative of the function $f$ is defined by the following expression:

$$D_q f(x) = \frac{d_q f(x)}{d_q x} = \frac{f(qx) - f(x)}{(q - 1)x}$$

Further derivatives can be defined in a manner analogous to their real counterparts. For example, the second $q$-derivative is defined as

$$D_q^2 f(x) = \frac{D_q [D_q f(x)]}{D_q x}$$

The standard rules for differentiation of products and quotients apply in quantum calculus:

$$d_q (f(x)g(x)) = g(x)d_q f(x) + f(x)d_q g(x)$$

$$D_q \frac{f(x)}{g(x)} = \frac{g(x)D_q f(x) - f(x)D_q g(x)}{g(x)^2}$$

To prove the dynamic programming algorithm, we will require the use of induction in the quantum calculus. Induction on time scales other than the integers must take into consideration the fact that successor and predecessors are not necessarily uniform. Therefore, use of the forward and backward jump operators is necessary. A form of induction holds true on time scales. If $t_0 \in \mathbb{T}$ and $S(t)$ is a statement for each $t \in [t_0, \infty)$ such that the following four conditions hold:

1. $S(t_0)$ is true
2. $S(t)$ being true at a right-scattered $t$ forces $S(\sigma(t))$ to be true
3. $S(t)$ being true at a right-dense $t$ forces $S(t')$ to be true for all $t'$ in a right-neighborhood of $t$
4. $S(t')$ being true for all $t' \in [t, t_0)$ when $t$ is right-dense forces $S(t)$ to be true

then it can be concluded that $S(t)$ is true for all $t \in [t_0, \infty)$. Further details can be found in [11]. For our purposes this basic overview will suffice. There is also a dual version which involves left-scattered and left-dense intervals and uses the backwards jump operator $\rho(t)$ to perform backwards inductive proofs. It is actually this dual version that we use in the proof of the Dynamic Programming Algorithm. Note further that for the $q$-time scale only conditions 1 and 2 need to be met since this time scale lacks dense points.

For the proof of ordered derivatives in the quantum calculus, we will need to employ differentiation with respect to a function. The Stieltjes integral provides a means for this in the traditional calculus, where we have the relation $\int f dg = \int f g^\prime dt$. We define a similar construction on the $q$-time scale where $\int f d\Delta = \int f g^\prime d\Delta$. Now, as the $q$-time scale is a fundamentally discrete set the integrals become expressed as summations as shown at the end of this section. We present this notation in the more general case to maintain consistency with the relationship between the $q$-calculus and general time scales where the formula is given by $\int f d\Delta = \int f g^\prime d\Delta$, where the symbol $\Delta$ denotes the idea of delta differentiation, which is the generalization of $q$-differentiation to any time scale. Deeper analysis of these concepts is beyond the scope of this paper; the interested reader is directed to [9], [12], and [19] for further information on the theory of integration on time scales.

Let $f: \mathbb{T} \to \mathbb{R}$ and $g: \mathbb{T} \to \mathbb{R}$ be functions on a $q$-time scale $\mathbb{T}$, and define the following $q$-derivative:

$$\frac{d_q f}{d_q g} = f^q g$$

This is simply the $q$-time scale analog of the expression $\frac{df}{dg} = \frac{df}{dt} \frac{dt}{dg}$ from classical analysis. We will make use of the $f^q g$ notation $f^q g$ in our proof of ordered derivatives on time scales.

Finally, a further note on the translation of integrals. Let $f: \mathbb{T} \to \mathbb{R}$ where $\mathbb{T} = \{t_0, t_1, \ldots\}$. Then

$$\int_{t_0}^{t_1} f(t) \Delta t = \sum_{i=t_0}^{t_1} f(t_i) \mu(t),$$

where $\mu(t)$ is the graininess function of the time scale $\mathbb{T}$. In particular, for the $q$-time scale we arrive at the following:

$$\int_{t_0}^{t} f(t) \Delta = \sum_{i=t_0}^{t} f(q^i) q^i (q - 1)$$

This construction is important in our discussions of dynamic programming and backpropagation, as it is convenient and illuminating to first consider the general time scale formulation in some cases before delving into the particulars of the quantum calculus version.

III. DYNAMIC PROGRAMMING IN THE QUANTUM CALCULUS

Dynamic programming concerns itself with the solution of multistate decision problems. Key elements include a set of decision points for making control choices, definitions of system states which capture all salient details of interest to the modeler, policies which map states to controls, a set of...
costs/rewards for each state/decision choice, a cost-to-go function measuring the costs of future action under a given policy, and a possible source of random disturbance to emulate the probability that the decision maker is operating under conditions of less than perfect information regarding the consequences of control selections upon state transitions. The reader unfamiliar with the above framework is directed to the texts [4], [5], [25], and [31] for further details.

We set out with the following definitions: decision points $t_t$ contained in a q-time scale $\mathbb{T}$ with a terminal point $T$, a set of controls for each state given by $c(x(t), t)$, random disturbances modeled by a stochastic term $w(t)$, a cost/reward function denoted by $r(x(t), c(x(t), t), w(t), t)$ with terminal point $T$ defined piecewise as $r_T(x(T))$, and a dynamic system where states $x(t)$ evolve according to the following rule:

$$d_t x(t) = f(x(t), c(x(t), t), w(t), t), \quad (7)$$

For policies $\pi$, we require each set of state-control pairs to represent a valid association for both the space and time dimensions. The tail of the policy $\pi$, denoted as $\pi^t$, chronicles the set of state-action pairs starting at decision point $t$ and ending at the terminal point $T$. This notion is critical for discussion of the optimality principle and the subsequent derivation of the dynamic programming algorithm.

The cost-to-go function is given by

$$J_\pi(x(t), t) = E\left\{ \int_t^T r(x(t), c(x(t), t), w(t), t) \Delta t \right\}$$

$$= (q-1) E \left\{ \sum_{q=1}^{q^T} r(x(q^t), c(x(q^t), q^t), w(q^t), q^t) \right\} \quad (8)$$

where the expansion in terms of q-time scales makes use of equation (6).

Before continuing, a brief aside on the nature of the expectation in the cost-to-go function is in order. In service of the desire to balance the dueling needs of robust modeling and mathematical tractability, we force a requirement of countability on the random disturbance term $w(t)$. This decision limits us from considering stochastic terms such as Brownian motion or Gaussian noise while still permitting a wide array of useful probabilistic models. For example, the standard discrete-time interpretation of $w(t)$ takes the form of a transition probability matrix $P$ whose entries $(i, j, k)$ indicate the chance the system evolves from state $i$ to state $j$ under control choice $k$. While this is perhaps the most widely used form of the system, we will proceed with the more general equation given by (8) in the interest of maximal mathematical abstraction and utility for our theorems.

Regardless of the nature of our disturbance terms, we define an optimal policy $\pi^*$ to be one which minimizes the cost-to-go functional $J$. The corresponding optimal cost-to-go functional is denoted

$$J^*(x(t_0), t_0) = \min_{\pi} J_\pi(x(t_0), t_0) \quad (9)$$

where the $\min$ is considered over all policies. The goal of the dynamic programming problem is to calculate an optimal policy $\pi^*$. The most basic process by which this is achieved is called the Dynamic Programming Algorithm.

This algorithm is a form of backwards induction. Starting from the terminal decision point $T$ and following a schedule of recursively defined steps backwards in time towards the initial point $t_0$, the optimal policy can be calculated even in a stochastically rich environment. The algorithm begins with setting $J(x(T), T) = r_T(x(T))$ and proceeds via the following rule:

$$J(x(t), t) = \min_{\pi} \left\{ r(x(t), c(x(t), t), w(t), t) + J(f(x(t), c(t), w(t), t), q^t) \right\} \quad (10)$$

This recursion is a consequence of Bellman’s Principle of Optimality, which states that any optimal policy must be still optimal when enacted on any tail of the system. That is, the solution which minimizes the cost-to-go function starting at any given point $t$,

$$(q - 1) E \left\{ \sum_{q=1}^{q^t} r(x(q^t), c(x(q^t), q^t), w(q^t), q^t) \right\} \quad (11)$$

is simply the portion of the optimal policy $\pi^*$ which coincides with the particular tail in question. The justification of this principle runs as a proof by contradiction: If the tail problem had a different solution than that given, then the cost-to-go function could be minimized further by changing out the optimal policy’s tail with this alternate policy, thus prohibiting the optimal policy from being, in fact, optimal. This cannot be the case so the optimality principle must hold. This concept is used in the proof of the dynamic programming algorithm given below.

Our proof follows that of [4] and suffices to establish the viability of dynamic programming in quantum calculus.

Theorem 1: The policy which minimizes the dynamic programming recursion (10) for all states and all times is optimal.

Proof: Set $J^*(x(T), T) = r_T(x(T))$ and proceed, via quantum calculus induction, to show that following the dynamic programming algorithm’s update rule yields the optimal policy each step of the way, i.e. that $J^*(x(t), t) = J(f(x(t), t))$ for all $t \in T$. Since the nature of this algorithm is to proceed backwards in time, we will use the dual version of time scales induction as described in section II of this paper. (Recall that the backwards jump operator $\rho(t) = q^{-1}$ for the q-time scale. However, we will maintain the use of the symbol $\rho(t)$ and trust no confusion will arise. This notation has the advantage of more closely mirroring the form of the version of the quantum calculus induction algorithm we invoke in the proof.)

Letting $t = T$ yields, by definition,

$$J^*(x(N), N) = r_T(x(T)) = J(x(N), N) \quad (12)$$
Now assume $J^*(x(t), t) = f(x(t), t)$ for some time point \( t \in T \) and all states \( x(t) \). Then we have

\[
J'(x(\rho(t)), \rho(t)) = r(x(t), c(x(t), t), w(t), \rho(t)) + \min \mathbb{E} (\mathbb{E}_{q} \left[ \sum_{i=q}^{T} r(x(q'), c(x(q'), \rho'), w(q'), q') \right])
\]

(14)

Note that the minimization is taken term by term over all controls and policies, respectively. We now use the principle of optimality to distribute the \( \min \) through the expectation, as the tail problem is an optimal policy for the sub-problem contained within the tail. This yields the following:

\[
J'(x(\rho(t)), \rho(t)) = \min \mathbb{E} (\mathbb{E}_{q} \left[ \sum_{i=q}^{T} r(x(q'), c(x(q'), \rho'), w(q'), q') \right])
\]

(15)

Using the definition of $J^*(x(t), t)$, which subsumes the term minimized over the policy, we can reduce this expression to

\[
J'(x(\rho(t)), \rho(t)) = \min \mathbb{E} (\mathbb{E}_{q} \left[ \sum_{i=q}^{T} r(x(q'), c(x(q'), \rho'), w(q'), q') \right]) + J(x(\rho(t)), t))
\]

By the induction hypothesis, we know the optimal cost-to-go is equivalent to the approximation due to the dynamic programming algorithm. Thus, we write

\[
J'(x(\rho(t)), \rho(t)) = \min \mathbb{E} (\mathbb{E}_{q} \left[ \sum_{i=q}^{T} r(x(q'), c(x(q'), \rho'), w(q'), q') \right]) + J(x(\rho(t)), t))
\]

which, by definition, is simply

\[
J'(x(\rho(t)), \rho(t)) = J(x(\rho(t)), t))
\]

(16)

Which, in turn, is our desired result.

With this, the dynamic programming algorithm is shown to work in quantum calculus. In fact, this can be interpreted as the Hamilton-Jacobi-Bellman equation in the quantum calculus, as the $q$-time scale is isolated. It should be noted, however, that this algorithm is quite computationally expensive, particularly for industrial-scale problems [23]. To circumvent this falling, suboptimal methods are routinely employed. Collectively called Approximate Dynamic Programming (ADP), these algorithms seek to calculate suboptimal policies to whatever degree of accuracy is required by a given application. These techniques are tied quite intimately to backpropagation, and hence the ordered derivative calculations, to tune $W$ to reduce an error measure between $\hat{Y}(t)$ and $Y(t)$. The implicit time scale is an isolated subset of $\mathbb{Z}$. Throughout this section we will consider time scales of higher generality and will assume $t \in T$, where $T$ is not restricted to a subset of $\mathbb{Z}$. We will let $T$ denote the number of inputs or the time the system is allowed to compute.

We define the following measure of predictive system error:

\[
E = \int_{1}^{\sigma(T)} E(t) \, \Delta \tau
= \int_{1}^{\sigma(T)} \sum_{i=1}^{n} \left( \frac{1}{2} (\hat{Y}(t) - Y(t))^2 \right) \, \Delta \tau
\]

(17)

This form represents the standard least-squares error measure in common use among statistical analysts. Note that the integral plays the part of the generalized summation operation and is appropriate for any time scale $T$ free of restriction to an isolated or, more specifically, quantum case. The proper summation that remains in the equation runs over the dimension of the output vector: a fixed scalar value unrelated to our time scale $T$ and thus not subsumed by an integral.

IV. ORDERED DERIVATIVES AND BACKPROPAGATION

To generalize from dynamic programming in quantum calculus to approximate dynamic programming we need to investigate backpropagation. This derivative calculation engine is often utilized in neural network training algorithms and figures prominently in ADP techniques ([23], [24], [29]). It is therefore of interest to demonstrate its validity on $q$-time scales. In his PhD dissertation ([36], reprinted in [34]), Paul Werbos introduced the notion of an ordered derivative as separate from derivatives of typical application in physical systems. The use of these ordered derivatives is motivated by the particular character of analysis of social systems as compared to a system operating under strictly physical dynamics. Relying on a series of ordered variables, each a consequence of the previous variables in the system, Werbos notes that the traditional total derivative chain rule fails to provide adequate calculations for investigation of such an integrated social system as well as for a wide array of other mathematically related connectionist systems, the model application given in Werbos’s dissertation being that of political forecasting. Werbos thereby establishes backpropagation as a viable manner in which to calculate derivatives not only in neural network training but for the manipulation of weights in a broad spectrum of adaptive systems. To complete the required calculations Werbos proved a special chain rule for these ordered derivatives that worked for the applications in which the traditional chain rule of continuous analysis broke down. Further discussion of ordered derivatives can be found in [33].

In this section we formulate the ordered derivative in quantum calculus and prove the chain rule Werbos derived obtains in this alternate environment. We conclude that the backpropagation approach to $q$-derivative calculation is as valid as the one for the classical derivative and, as such, neural network training on quantum calculus may follow its traditional counterpart.

Following [35], we define the following system variables:

- input $X(t)$, output $\hat{Y}(t)$ which approximates the target output $Y(t)$.
- The relationship between inputs and outputs is determined via a relationship between inputs and outputs.
- The implicit time scale is an isolated subset of $\mathbb{Z}$.
- Throughout this section we will consider time scales of higher generality and will assume $t \in T$, where $T$ is not restricted to a subset of $\mathbb{Z}$. We will let $T$ denote the number of inputs or the time the system is allowed to compute.

We define the following measure of predictive system error:

\[
E = \int_{1}^{\sigma(T)} E(t) \, \Delta \tau
= \int_{1}^{\sigma(T)} \sum_{i=1}^{n} \left( \frac{1}{2} (\hat{Y}(t) - Y(t))^2 \right) \, \Delta \tau
\]

(17)
Our purposes require investigation of a specific time scale. Using equation (6), we can translate (17) into the terms of quantum calculus as follows:

\[
E = \int_{x}^{q} E(t) \Delta t \\
= \sum_{i=0}^{n-1} E(q^i) q'(q^i - 1) \\
= (q - 1) \sum_{i=0}^{n-1} a_i^n (Y(q^i) - Y(q^i))^2. 
\] (18)

Further analysis of the network equations requires the chain rule for ordered derivatives, as this is the tool used to transform the network equations into a calculator of the proper differentiation operator. In particular, the chain rule for quantum calculus hinges on sifting the network equations through the ordered derivatives. Let \( x_1(t), x_2(t), \ldots, x_{n}(t) \) be an ordered sequence of variables with \( t \in \mathbb{T} \). These variables represent stages of a larger calculation (e.g., layers, in a sense, of a multi-layer perceptron network) and follow a recursion given by

\[
x_1 = f_i(x_{i-1}, x_{i-2}, \ldots, x_1)
\]

so that we can speak meaningfully of causation as a basis for the relationships among the \( x_i \)'s. We are interested in determining the way the error \( E \) changes with respect to one of the \( x_i \)'s, i.e. we want to calculate \( \frac{dE}{dx_i} \) or, using the notation from the preliminaries section, we want to calculate \( E^dq^{x_i} \). Following [34], we set up the error as a sequence of recursive functions such that

\[
E_0(x_0, \ldots, x_1) = x_n
\] (19)

and

\[
E_{i-1}(x_{i-1}, \ldots, x_1) = E_i(f_i(x_{i-1}, \ldots, x_k), x_{i-1}, \ldots, x_1). 
\] (20)

Then, the ordered derivative of \( E \), which equals \( x_n \) via our construction, is defined to be

\[
E^d q^{x_i} = \frac{dE}{dx} = E^d q^{x_i}.
\] (21)

The backpropagation algorithm of derivative calculation hinges on sifting the network equations through the ordered differentiation operation. In particular, the chain rule for ordered derivatives plays a key role. The following theorem establishes this chain rule for quantum calculus.

**Theorem 2:** \( \frac{dE}{df_j} = \sum_{k=j+1}^{n} x_n^d x_k f_k \frac{dE}{dx_k} \)

**Proof:** As in [34], we proceed by induction on \( j \). We will start with \( j = n - 1 \) and end with \( j = i \). With \( j < i \), the recursive definitions of the \( f_i \)'s and \( x_i \)'s force the terms \( f_k^d q^{x_i} \) to zero and, therefore, they do not contribute to the summation. So, it will suffice to consider \( j \) in the range \( n - 1 \) to \( i \).

Let \( j = n - 1 \). Then our hypothesis becomes

\[
E^d q^{x_i} = x_n^d x_{n+1} f_{n+1} = x_n^d x_n f_n.
\] (22)

Calling on the definition of the sequence of \( E_i \)'s, we see that \( E^d q^{x_i} = f_n^d x_n \) so that, since \( x_n^d q^{x_i} = 1 \), the claim is proven.

Now assume the hypothesis is true for some \( j + 1 < n \). Our task is to show the claim holds for \( j > i \). Consider \( dE_j f_j q^{x_i} = E_j^d q^{x_i} \). Since \( E \) is defined from \( \mathbb{R} \to \mathbb{R} \), the delta derivative construction reduces to the traditional case. Also, by definition, \( E_j(f_j, \ldots, x_1) = E_j(f_j+1, \ldots, x_1) \).

Therefore,

\[
E^d q^{x_i} = \sum_{f_i}^d q^{x_i} = f_{j+1}^d x_{j+1} dE_{j+1} + dE_{j+1} f_{j+1} q^{x_i} + \ldots + dE_{j+1} f_{j+1} q^{x_i} + \ldots
\] (23)

for \( i \leq j \). From our definition from the preliminaries applied to our recursive definition of the ordered variables \( x_i \), we have that \( dE_j f_j q^{x_i} = 0 \) when \( k < i \), as the preceding variables in the order are unaffected by the later variables in the causation chain. This result allows us to reduce our equation to

\[
E^d q^{x_i} = \sum_{f_i}^d q^{x_i} + \ldots + \sum_{f_i}^d q^{x_i} + \ldots
\] (24)

which is our desired result.

Thus, the chain rule for ordered derivatives in quantum calculus is established. With this result, we are able to construct neural network architectures in quantum calculus and train them via backpropagation. While the traditional chain rule of classical analysis fails to hold for ordered derivatives, the chain rule for ordered derivatives does hold on \( f \)-time scales. Since time scales in general, and \( f \)-time scales in particular, may be the appropriate mathematical framework to discuss a certain class of resource allocation problems, and dynamic programming concerns itself with optimization of multi-stage decision scenarios, a quantum calculus approach to the approximation of the optimal solution becomes an exciting new area of computational decision theory.

3694 \textbf{2008 International Joint Conference on Neural Networks (IJCNN 2008)
V. CONCLUSION AND EXTENSIONS

The calculus of time scales in general, and the q-time scale studied in the quantum calculus in particular, is an increasingly relevant and emerging area of mathematics with wide-ranging opportunities for application. We have established the functionality of ordered derivatives in the quantum calculus, and by doing so extended backpropagation to this domain. Furthermore, we showed the dynamic programming algorithm derived from Bellman’s principle of optimality, also obtains on q-time scales. Together, backpropagation and the dynamic programming algorithm form the foundation of ADP. As such, we have connected two major fields of research.

It is important to note that many of the ideas in dynamic programming, including the Bellman equation itself, have their birth in the calculus of variations ([15], [26], [37]) which is just now beginning its translation into the language of time scales ([6], [8], [14]). While the calculus of variations is now considered to be a proper subset of functional analysis, a further investigation into dynamic programming in the quantum calculus may do well to take into account classical variational principles. This stord domain still has much to offer researchers in optimization.

In particular, and in keeping with the place held by the q-calculus in physics, it is interesting to investigate quantum mechanical extensions of the Hamilton-Jacobi-Bellman equation for use in dynamic programming. While there is a necessary limit to the similarities between the motion of a particle in a physical system and the control path of an agent in a multi-stage decision process, we believe that this limit has yet to be reached under the current Hamilton-Jacobi-Bellman theoretics. The impact of the quantum calculus on computational decision theory may not have yet reached its peak, and we believe further thought in these directions can yield great rewards.

REFERENCES