Fourier–Hermite Dynamic Programming for Optimal Control
Syeda Sakira Hassan, Member, IEEE, and Simo Särkkä, Senior Member, IEEE

Abstract—In this article, we propose a novel computational method for solving nonlinear optimal control problems. The method is based on the use of Fourier–Hermite series for approximating the action-value function arising in dynamic programming instead of the conventional Taylor-series expansion used in differential dynamic programming. The coefficients of the Fourier–Hermite series can be numerically computed by using sigma-point methods, which leads to a novel class of sigma-point-based dynamic programming methods. We also prove the quadratic convergence of the method and experimentally test its performance against other methods.

Index Terms—Approximate dynamic programming, differential dynamic programming, Fourier–Hermite series, sigma-point dynamic programming, trajectory optimization.

I. INTRODUCTION

Trajectory optimization in nonlinear systems is an active research area in optimal control and reinforcement learning [1], [2], [3]. The aim is to find a state-control sequence that globally or locally minimizes a given performance index such as a cost or a reward function. Applications include trajectory planning in autonomous vehicles, robotics, industrial automation, and gaming [4], [5], [6], [7], [8], [9].

A commonly used approach for solving trajectory optimization problems is dynamic programming (DP) [2], [10], which is based on solving the value function from the Bellman’s equation [10] by using suitable numerical methods. One such particular approach is differential dynamic programming (DDP) [11], [12], [13], where a locally optimal solution is reached iteratively by backward and forward passes. The method is based on the second-order Taylor series expansion of the action-value function that appears in the Bellman’s equation of dynamic programming. The convergence of DDP has also been proven under suitable differentiability conditions [14], [15], [16].

Although DDP has turned out to be useful in many applications, the second-order Taylor series expansion used in this method is computationally expensive due to the higher order derivatives appearing in the expansion. Therefore, researchers have opted to discard the second-order derivatives, which has led to methods such as the iterative linear quadratic regulator (iLQR) [17]. Furthermore, Taylor series expansion is also an inherently local approximation as it is based on derivatives evaluated at a single point and it induces strong differentiability assumptions on the dynamics and cost functions [14], [15], [16]. To address these limitations, the Taylor series expansion can also be replaced with other approximations. Examples of such methods are the unscented DP [18], sparse Gauss–Hermite quadrature DDP [19], and sampled DDP [20].

II. DIFFERENTIAL DYNAMIC PROGRAMMING

In this section, we define the control problem to be solved and review the differential dynamic programming (DDP) algorithm.

A. Problem Formulation

Consider a nonlinear discrete-time deterministic optimal control problem [2] with cost

$$J(u_{1:T-1};x_1) = \ell_T(x_T) + \sum_{k=1}^{T-1} \ell_k(x_k, u_k)$$

with given initial state $x_1$, subject to the dynamics of the form

$$x_{k+1} = f_k(x_k, u_k), \quad k = 1, \ldots, T - 1. \quad (2)$$

Here, $x_k \in \mathbb{R}^n$ is the state variable, $u_k \in \mathbb{R}^s$ is the control variable at step $k$, and $u_{1:T-1} = \{u_1, \ldots, u_{T-1}\}$ is a sequence of controls over the horizon $T$. For a given initial state $x_1$, the total cost of the control sequence $u_{1:T-1}$ is given by (1). Furthermore, $\ell_T(x_T)$ denotes the terminal cost of the state $x_T$ and the $\ell_k(x_k, u_k)$ is the cost incurred at time step $k$. In particular, unscented DP [18] uses an unscented transform-based method, inspired by the unscented Kalman filter [21], [22], to estimate the derivatives using a sigma-point scheme. This allows the DP algorithm to be derivative-free while leveraging information beyond a single point of evaluation and without compromising the performance of the classical DDP algorithm. Additionally, cubature approximations of stochastic continuous-time DDP are considered in [23] and probabilistic approximations based on Gaussian processes are considered in [24].

The contribution of this article is to propose a method based on Fourier–Hermite series (cf. [25]) to approximate the action-value function. The resulting Fourier–Hermite dynamic programming (FHDP) algorithm can be implemented using sigma-point methods in a completely derivative-free manner, which leads to a new class of sigma-point dynamic programming (SPDP) methods. We also prove the local second-order convergence of the method and experimentally evaluate its performance against classical DDP and unscented DP. Unlike unscented DP or sparse Gauss–Hermite DDP, the method is guaranteed to converge in well-defined conditions, and it can also explicitly handle nonquadratic costs. Moreover, unscented DP requires the propagation of estimates in backward direction along the trajectory, which is not needed in our method.

The article is structured as follows. In Section II, we revisit the DDP in discrete-time domain. In Section III, we first discuss Fourier–Hermite series and then use the Fourier–Hermite expansion to approximate the action-value function, leading to the proposed method. In Section IV, we analyze the computational complexity and prove the local convergence of the method, and in Section V we experimentally evaluate its performance. Concluding remarks are given in Section VI.
The aim is to find a control sequence $u_{1:T-1}$ that minimizes the cost defined by (1)

$$u_{1:T-1} = \arg\min_{u_{1:T-1}} J(u_{1:T-1}; x_1).$$

(3)

This solution can be expressed in terms of the optimal cost-to-go or value function $V_k(x_k)$ that gives the minimum total cost accumulated between time step $k$ and $T$ starting from the state $x_k$. As shown by Bellman [10], we can compute the value function using backward recursion as follows:

$$V_k(x_k) = \min_{u_k} \{ \ell_k(x_k, u_k) + V_{k+1}(f_k(x_k, u_k)) \}$$

(4)

where the value function at the terminal time $T$ is $V_T(x_T) = \ell_T(x_T)$.

Because (4) becomes computationally infeasible with increasing state-dimensionality [26], one common approach is to approximate the action-value function appearing on the right-hand side of (4) using a tractable approximation. In particular, DDP, which is discussed below, uses a second-order Taylor series expansion for this purpose.

### B. Differential Dynamic Programming

The classical DDP [11], [12], [13] approach uses a second-order Taylor series expansion of the action-value function $Q_k$ about a nominal trajectory. Given a nominal trajectory of states and controls $(\hat{x}_k, \hat{u}_k)$, at step $k$, we can approximate $Q_k$ around this trajectory using a second-order Taylor series expansion

$$Q_k(x_k, u_k) \approx Q_k^0 + Q_k^1 \delta x_k + Q_k^2 \delta u_k$$

where $\delta x_k = x_k - \hat{x}_k$ and $\delta u_k = u_k - \hat{u}_k$. Let us now approximate a quadratic for the value function of the form

$$V_{k+1}(f_k(x_k, u_k)) \approx V_{k+1}^0 - v_k^T \delta x_{k+1} + \frac{1}{2} \delta x_{k+1}^T S_{k+1} \delta x_{k+1}.$$  

(7)

If we now form a second-order Taylor series expansion of (5), we get the following coefficients for (6):

$$Q_k^0 \approx \ell_k(\hat{x}_k, \hat{u}_k) + V_{k+1}^0 - v_k^T \delta x_{k+1} (f_k(\hat{x}_k, \hat{u}_k) - \hat{x}_{k+1})$$

$$+ \frac{1}{2} (f_k(\hat{x}_k, \hat{u}_k) - \hat{x}_{k+1})^T S_{k+1} (f_k(\hat{x}_k, \hat{u}_k) - \hat{x}_{k+1})$$

$$Q_k = L_x + F_x^T [v_k - v_{k+1} + S_{k+1} (f_k(\hat{x}_k, \hat{u}_k) - \hat{x}_{k+1})]$$

$$Q_{xx} = L_{xx} + F_{xx}^T [v_k - v_{k+1} + S_{k+1} (f_k(\hat{x}_k, \hat{u}_k) - \hat{x}_{k+1})]$$

$$Q_{uu} = L_{uu} + F_{uu}^T [v_k - v_{k+1} + S_{k+1} (f_k(\hat{x}_k, \hat{u}_k) - \hat{x}_{k+1})]$$

(8)

The second-order derivative matrices of the $m$th component of $f_k$. All the derivatives are evaluated at $(\hat{x}_k, \hat{u}_k)$.

Minimizing (6) with respect to $\delta u_k$, we arrive at the following correction to the control trajectory:

$$\delta u_k = -Q_{uu}^{-1} Q_{u\delta x_k}^T \delta x_k.$$  

(9)

Let us define

$$d = -Q_{uu}^{-1} Q_{u\delta x_k}, \quad K = Q_{uu}^{-1} Q_{ux}.$$  

(10)

then we can rewrite (9) as follows:

$$\delta u_k = d - K \delta x_k.$$  

(11)

By substituting $\delta u_k$ to (6), we get the coefficients for the quadratic approximation of the value function at step $k$:

$$V_k^0 = Q_k^0 + \frac{1}{2} d^T Q_u$$

$$v_k = -Q_{ux} - K^T Q_{uu} d$$

$$S_k = Q_{xx} - K^T Q_{uu} K.$$  

(12)

This procedure is then continued backwards for $k - 1, k - 2, \ldots, 1$.

That is, the backward pass of DDP starts from the terminal time step $k = T$ from a quadratic approximation to $\ell_T$ formed with a second-order Taylor series expansion centered at $\hat{x}_T$. Then, we successively perform the aforementioned computations until $k = 1$. The backward pass is followed by a forward pass, where the system is simulated forward in time under the optimal control law (11) to generate a new trajectory. The backward and forward passes are iterated until convergence. The pseudocode for the classical DDP method is given in Algorithm 1.

### C. Regularization of the Optimization and Line Search

As with all nonlinear optimization, proper care must be taken to ensure a good convergence behavior of the method. The DDP algorithm involves the matrix inversion of $Q_{uu}$ in (9), which may cause numerical instability. A regularization scheme was, therefore, proposed by [12], [16], [27] to ensure invertibility of $Q_{uu}$ in (9) by replacing it with

$$\tilde{Q}_{uu} = Q_{uu} + \beta I$$  

(13)

The algorithm starts from the terminal time step $T$ and proceeds backwards, estimating a quadratic approximation to $\ell_T$ at each time step. The algorithm iterates between the forward and backward passes until convergence.
where $\beta > 0$ is a small positive constant. Furthermore, when using this regularization, Tassa [27] suggests that the following modifications to (12) are recommended for numerical stability:

$$V_0^k = Q_i^k + \frac{1}{2} d^\top Q_u + \frac{1}{2} d^\top Q_{uu} d$$

$$v_k = -Q_x + \tilde{K}^\top Q_{uu} d + \tilde{K}^\top Q_u - Q_{ux}^u d$$

$$s_k = Q_{xx} + \tilde{K}^\top Q_{uu} \tilde{K} - \tilde{K}^\top Q_u - Q_{ux}^u \tilde{K}$$

(14)

where $\tilde{K} = \tilde{Q}^{-1} Q_u$ and $d = \tilde{Q}^{-1} Q_x$.

The value $\beta$ can be adapted by using a Levenberg–Marquardt type of adaptation procedure, that is, if the cost of the new trajectory is less than the current one, then the value of $\beta$ is decreased by dividing it with a constant factor $\nu > 1$, otherwise the value increased by multiplying with $\nu$ and the new trajectory is discarded. Note that if $\beta = 0$, then (14) reduces to (12). In [28], Tassa et al. used a quadratic modification schedule to choose $\beta$ at each iteration.

The work in [15], [27], and [29] also suggested to use an additional backtracking line search scheme to improve the convergence. A parameter $0 < \epsilon < 1$ is introduced in the update statement of the control in the forward pass routine as follows:

$$\delta u_k = \epsilon \delta - \tilde{K} \delta x_k, \quad u_k = u_k + \delta u_k.$$  

(15)

We start by setting $\epsilon = 1$ and update control using (15) and then generate a new state sequence by forward simulation, that is, $\tilde{x}_{k+1} = f(\tilde{x}_k, \tilde{u}_k)$. If the decrease in the cost function is not below a given threshold, the value of $\epsilon$ is decreased (e.g., by halving it as we did in our case), and we restart the forward pass again.

D. Implementation of DDP Using Automatic Differentiation (AD)

During the backward pass in DDP, we need to evaluate the derivatives on the right-hand side of (8) at $(\tilde{x}_k, \tilde{u}_k)$ to approximate $Q_k$. Classically, these derivatives have been derived by hand or via symbolic or numerical differentiation methods, but they can also be automatically computed by using AD [30]. AD is based on transforming the function to be evaluated into a sequence of operations that compute the exact derivatives of the function along with its value. AD is readily available in several programming platforms such as TensorFlow [31], PyTorch [32], and MATLAB [33].

When using AD, there are two alternative ways to evaluate $Q_k$ of the DDP algorithm at the nominal trajectory $(\tilde{x}_k, \tilde{u}_k)$. The first approach evaluates the derivatives of $\ell_k$ and $f_k$ on the right-hand side of (8) at $(\tilde{x}_k, \tilde{u}_k)$ using AD. Once we have all the derivatives, we can solve for $Q_k$ using (8). The alternative is to use AD directly to compute $Q_k$ and evaluate its derivatives at $(\tilde{x}_k, \tilde{u}_k)$. In this article, we implement DDP with AD by applying the former approach due to its more direct connection with the classical DDP.

III. FOURIER–HERMITE DYNAMIC PROGRAMMING

In this section, we present the proposed method which is based on using the Fourier–Hermite series approximation instead of the Taylor series approximation for the action-value function.

A. Fourier–Hermite Series

Fourier–Hermite series is a series expansion of a function using Hermite polynomial basis of a Hilbert space [34]. The nth order univariate Hermite polynomials can be computed as follows:

$$H_m(x) = (-1)^m \exp(x^2/2) \frac{d^m}{dx^m} \exp(-x^2/2), \quad m = 0, 1, \ldots,$$

(16)

The first few $(m = \{0, 1, 2\})$ Hermite polynomials are $H_0(x) = 1$, $H_1(x) = x$, $H_2(x) = x^2 - 1$, and for $m \geq 3$ polynomials can be found using the recursion $H_{m+1}(x) = x H_m(x) - m H_{m-1}(x)$.

A multivariate Hermite polynomial with multindex $I = \{i_1, \ldots, i_n\}$ for n-dimensional vector $x$ can be written as

$$H_I(x) = \prod_{i=1}^{n} H_{i,m}(x_{m})$$

(17)

where $H_{i,m}(x_{m})$ are univariate Hermite polynomials. Let us define the inner product of two functions $f$ and $g$ as

$$\langle f, g \rangle = \int_{\mathbb{R}^n} f(x) g(x) N(x \mid 0, I) dx$$

(18)

and a Hilbert space $\mathcal{H}$ consisting of functions satisfying $\|g\|^2 = \langle g, g \rangle < \infty$. Then, the Hermite polynomials are orthogonal in the sense

$$\langle H_I, H_J \rangle = \int H_I(x) H_J(x) N(x \mid 0, I) dx = \begin{cases} \mathbb{I}, & \text{if } I = J, \\ 0, & \text{otherwise}. \end{cases}$$

(19)

Here, $\mathbb{I} = i_1! i_2! \cdots i_n!$, $J = \{j_1, j_2, \ldots, j_n\}$ and $I = J$ when $i_k = j_k$ for all elements in $I, J$. Now, we can define the Fourier–Hermite expansion of a function $g(x)$ as follows.

Definition 1: For any $g \in \mathcal{H}$, the Fourier–Hermite expansion of $g$ with respect to a unit Gaussian distribution $N(0, I)$ is given by

$$g(x) = \sum_{k=0}^{\infty} \sum_{|I|=k} \frac{1}{k!} c_I H_I(x)$$

(20)

where $H_I$ is a multivariate Hermite polynomial and $c_I$ are the series coefficients given by the inner product $c_I = \langle g, H_I \rangle$.

The representation in (20) is useful if we want to compute expectations of a nonlinear function over a unit Gaussian distribution. It turns out that the expectation of the function can be simply extracted from the zeroth order coefficient $c_0$ of the Fourier–Hermite series and the higher order coefficients are equal to the expectations of the derivatives of the function $g(x)$ [25]. In this article, we are particularly interested in the second-order Fourier–Hermite series expansion of $g(x)$, which can be written as

$$g(x) \approx \sum_{k=0}^{2} \sum_{|I|=k} \frac{1}{k!} c_I H_I(x) = \mathbb{E}[g(x)] + \mathbb{E}[g(x) H_1(x)]^\top H_1(x)$$

(21)

$$+ \frac{1}{2} \text{tr} \left[ \mathbb{E}[g(x) H_2(x)] H_2(x) \right].$$

In (21), the multivariate polynomials $H_1(x)$ have been expressed as vectors and matrices as follows (cf. [25]):

$$H_0(x) = 1, \quad H_1(x) = x, \quad H_2(x) = xx^\top - I.$$  

(22)

We can also generalize the expansion to a more general Gaussian distribution $N(\mu, \Sigma)$ by rewriting the second order Fourier–Hermite expansion as

$$g(\Lambda x + \mu) \approx \mathbb{E}[g(\Lambda x + \mu)] + \mathbb{E}[g(\Lambda x + \mu) H_1(x)]^\top H_1(x)$$

$$+ \frac{1}{2} \text{tr} \left[ \mathbb{E}[g(\Lambda x + \mu) H_2(x)] H_2(x) \right].$$

(23)

Above, we have put $\Sigma = \Lambda \Lambda^\top$. If we now let $y = \Lambda x + \mu$, then (23) becomes

$$g(y) \approx \mathbb{E}[g(y)] + \mathbb{E}[g(y) H_1(\Lambda^{-1}(y - \mu))]^\top H_1(\Lambda^{-1}(y - \mu))$$

$$+ \frac{1}{2} \text{tr} \left[ \mathbb{E}[g(y) H_2(\Lambda^{-1}(y - \mu))] H_2(\Lambda^{-1}(y - \mu)) \right].$$

(24)

where the expectations are over $y \sim N(\mu, \Sigma)$. Now, if we substitute the multivariate Hermite polynomials from (22) to (24), we get

$$g(y) \approx \mathbb{E}[g(y)] + \mathbb{E}[g(y) H_1(\Lambda^{-1}(y - \mu))]^\top (\Lambda^{-1}(y - \mu))$$

$$+ \frac{1}{2} \text{tr} \left[ \mathbb{E}[g(y) H_2(\Lambda^{-1}(y - \mu))] (\Lambda^{-1}(y - \mu)) \right].$$

(25)
IEEE TRANSACTIONS ON AUTOMATIC CONTROL, VOL. 68, NO. 10, OCTOBER 2023

is the joint covariance of the nominal trajectory

\[ \int (\Lambda^{-1} (y - \mu) (y - \mu)^T \Lambda^{-1} - I) \right\}. \quad (25) \]

Let us denote

\[ a_G = \mathbb{E} [g(y)] \]
\[ b_G = \mathbb{E} [g(y) H_1 (\Lambda^{-1} (y - \mu))] \]
\[ C_G = \mathbb{E} [g(y) H_2 (\Lambda^{-1} (y - \mu))]. \quad (26) \]

Then, (25) can be rewritten as

\[ g(y) \approx a_G - \frac{1}{2} \text{tr} \{ C_G \} + b_G (\Lambda^{-1} (y - \mu)) \]
\[ + \frac{1}{2} (y - \mu)^T [\Lambda C_G^{-1} \Lambda^{-1}] (y - \mu). \quad (27) \]

Now, we are ready to apply this approximation to the action-value function.

B. Fourier–Hermite Approximation of Action-Value Function

Consider the approximation of the action-value function \( Q_k \) defined in (5). Furthermore, assume that we have a quadratic approximation to the value function at step \( k + 1 \) of the form (7). Instead of using a Taylor series approximation to get (6) as in (8), we will now form the approximation with Fourier–Hermite series. Assume that our nominal trajectory for \( k = 1, \ldots, T - 1 \) consists of mean \( \mu_k = [\bar{x}_k, \bar{u}_k] \) and the joint covariance \( \Sigma_k = \Lambda_k \Lambda_k^T \) for the Gaussian distribution of the state-control pair \( (x_k, u_k) \). Further assume that at the terminal step \( T \) the nominal trajectory consists of \( \bar{x}_T \) and \( \Sigma_T = \Lambda_T \Lambda_T^T \). If we let \( \delta x_k = x_k - \bar{x}_k \) and \( \delta u_k = u_k - \bar{u}_k \), then the Fourier–Hermite approximation for \( Q_k \) can be written as

\[ Q_k(x_k, u_k) \approx a_Q - \frac{1}{2} \text{tr} \{ C_Q \} + b_Q \Lambda_k^{-1} \left[ \delta x_k \right] \left[ \delta u_k \right] \]
\[ + \frac{1}{2} \left[ \delta x_k^T \delta u_k \right] \left[ \Lambda_k C_Q^{-1} \Lambda_k \right]^{-1} \left[ \delta x_k \right] \left[ \delta u_k \right]. \quad (28) \]

where

\[ a_Q = \mathbb{E} [Q_k(x_k, u_k)], \]
\[ b_Q = \mathbb{E} [Q_k(x_k, u_k) H_1 (\Lambda_k^{-1} \delta x_k, \delta u_k)] \]
\[ C_Q = \mathbb{E} [Q_k(x_k, u_k) H_2 (\Lambda_k^{-1} \delta x_k, \delta u_k)]. \quad (29) \]

with the expectations taken over the joint Gaussian distribution for \( (x_k, u_k) \). The expectations can be computed using numerical integration methods such as sigma-point methods [22]. By matching the terms in (6) and (28), we now notice that this approximation has the same form as DDP with the correspondences

\[ \begin{bmatrix} Q_k^0 \\ Q_k \end{bmatrix} = \begin{bmatrix} a_Q - \frac{1}{2} \text{tr} \{ C_Q \} \\ b_Q \Lambda_k^{-1} \end{bmatrix} \]
\[ \begin{bmatrix} Q_{x} \\ Q_{u} \end{bmatrix} = \left[ \Lambda_k C_Q^{-1} \Lambda_k \right]^{-1}. \quad (30) \]

At the terminal step \( T \), the nominal trajectory consists of mean \( \bar{x}_T \) and covariance \( \Sigma_T = \Lambda_T \Lambda_T^T \). The approximation is formed as

\[ V_T(x_T) \approx v_T^0 - v_T^T \delta x_T + \frac{1}{2} \delta x_T S_T \delta x_T \quad (31) \]

where \( \delta x_T = x_T - \bar{x}_T \), and

\[ v_T^0 = a_T - \frac{1}{2} \text{tr} \{ C_T \}, \]
\[ v_T^T = -b_T \Lambda_T^{-1}, \]
\[ S_T = [\Lambda_T C_T^{-1} \Lambda_T^T]^{-1}. \quad (32) \]

with

\[ a_T = \mathbb{E} [\ell_T(x_T)], \]
\[ b_T = \mathbb{E} [\ell_T(x_T) H_1 (\Lambda_T^{-1} \delta x_T)], \]
\[ C_T = \mathbb{E} [\ell_T(x_T) H_2 (\Lambda_T^{-1} \delta x_T)]. \quad (33) \]

The FHDP algorithm in its abstract form now consists in replacing the Taylor series based computations of the action-value function coefficients in (6) with (30) and the terminal step value function approximation by (32). It is worth noticing that the Hermite polynomials needed at the terminal step in (33) are functions of \( n \)-dimensional input although in (29) the input dimension is \( s + n \).

C. Coefficient Computation via Sigma-Point Methods

Sigma-point methods are numerical integration methods commonly used in nonlinear filters, such as unscented Kalman filters (UKFs), cubature Kalman filters (CKFs), Gauss–Hermite filters (GHF) [22]. In their most common form, sigma-point method can be seen as Gaussian quadrature approximations for computing Gaussian integrals as follows:

\[ \int \mathcal{N}(x \mid 0, I) dx \approx \sum_i W_i^n g(\xi_i^n) \quad (34) \]

where \( x \in \mathbb{R}^n \), and the weights \( W_i^n \) and (unit) sigma points \( \xi_i^n \) for the \( n \)-dimensional integration rule are determined by the sigma-point method at hand. By a change of variables, \( y = \Lambda x + \mu \) we can then approximate integrals over more general Gaussian distributions \( \mathcal{N}(y \mid \mu, \Sigma) \) as

\[ \int \mathcal{N}(y \mid \mu, \Sigma) dx \approx \sum_i W_i^n g(\Lambda \xi_i^n + \mu) \quad (35) \]

with \( \Sigma = \Lambda \Lambda^T \). We can now apply this rule to (29), which gives the sigma-point approximations:

\[ a_Q \approx \sum_i W_i^{n+s} Q_k(\Lambda_k \xi_i^{n+s} + [\bar{x}_k, \bar{u}_k]) \]
\[ b_Q \approx \sum_i W_i^{n+s} Q_k(\Lambda_k \xi_i^{n+s} + [\bar{x}_k, \bar{u}_k]) \xi_i^{n+s} \]
\[ C_Q \approx \sum_i W_i^{n+s} Q_k(\Lambda_k \xi_i^{n+s} + [\bar{x}_k, \bar{u}_k]) (\xi_i^{n+s} [\xi_i^{n+s}]^T - I) \quad (36) \]

where \( \Sigma_k = \Lambda_k \Lambda_k^T \) is the joint covariance of the nominal trajectory \((x_k, u_k)\). It is though important to note that it is not sufficient to use a third-order rule such as unscented transform or third-order cubature rule, because the resulting integrals are typically higher order polynomials. Instead, it is advisable to use, for example, Gauss–Hermite rules [35], [36] or higher order spherical cubature (i.e., unscented) rules [37], [38], [39].

The expectations at the terminal step (33) can be computed as

\[ a_T \approx \sum_i W_i^n \ell_T(\Lambda_T \xi_i^n + x_T), \]

1With a slight abuse of the notation \([x_k, u_k]\) used here to represent a column vector and is equivalent to \([x_k, u_k]^T\), not to be confused with \([x_k, u_k]\), which represents a row vector.
Algorithm 2: Sigma-Point Dynamic Programming (SPDP).

Input: Initial state \( \hat{x}_1 \), nominal control \( \hat{u}_k \), for \( k = 1, \ldots, T - 1 \), nominal state \( \hat{x}_k \), for \( k = 2, \ldots, T \), terminal covariance \( \Sigma_T \), and joint covariance \( \Sigma_k \).

Output: Update \( \hat{u}_k \) and \( \hat{x}_k \).

1: **Backward pass:**
2: Given \( \Sigma_T \), compute \( a_T, b_T, \) and \( C_T \) using (37).
3: Compute \( v_T, v_T, \) and \( S_T \) using (32).
4: for \( k = T - 1 \) to 1 do
5: Given \( \Sigma_k \), compute \( a_Q, b_Q \) and \( C_Q \) using (36).
6: Compute the coefficients of \( Q_k(x_k, u_k) \) using (30).
7: Compute \( d_i \) and \( K \) using (10), and \( v_k \) and \( S_k \) using (12).
8: end for
9: **Forward pass:**
10: Start from \( \hat{x}_1 \).
11: for \( k = 1 \) to \( T - 1 \) do
12: \( \hat{u}_k \leftarrow u_k + \delta u_k \), where \( \delta u_k \) is given by (11).
13: \( \hat{x}_{k+1} \leftarrow \xi_k(\hat{x}_k, \hat{u}_k) \).
14: end for
15: Repeat from Step 1 until convergence.

\[
\begin{align*}
    b_T & \approx \sum_i W_i^n \ell_i(\Lambda_T \xi^n_i + \hat{x}_T) \xi^n_i \\
    C_T & \approx \sum_i W_i^n \ell_i(\Lambda_T \xi^n_i + \hat{x}_T) \left( \xi^n_i \xi^n_i^\top \right)^{-1} - I.
\end{align*}
\]

The sigma-point-based FHDP is summarized in Algorithm 2. Although the algorithm is written in its simple form, it is also possible to use the regularization and line search methods described in Section II-C as part of it. Although in the line search, a straightforward way is to use the original cost function as the merit function, in its implementation, it is important to take into account that the cost function minimized by the FHDP is not exactly the original cost function (see Section IV).

IV. Theoretical Results

In this section, we discuss the computational complexity of DDP and sigma-point FHDP methods, and prove the local convergence of our proposed method.

A. Computational Complexity

Let us assume that the dimension of the state \( n \) dominates the dimension of the control \( s \). When implemented in form (8), the computational complexity of DDP in terms of function evaluations nominally depend on the complexity of evaluating the first-order and the second-order derivatives of the dynamics. Each of the first-order derivatives in (6) require \( O(n^2) \) operations. The second-order derivatives require \( O(n^3) \) operations. Therefore, the computational complexity of the DDP method per iteration is \( O(Tn^3) \), where \( T \) is the time horizon [16]. However, in some cases, we can decrease the complexity of DDP to \( O(Tn^2) \) [40] by using AD directly on \( Q_k \) as described in Section II-D.

In sigma-point methods, the computational complexity in terms of number of function evaluations is equal to the number of sigma-points. For instance, in Gauss–Hermite quadrature rule with order \( p \), the number of required sigma-points is \( p^n \), where \( m = n + s \). In this rule, the number of evaluation points grows exponentially with the state and control dimensions. On the other hand, in the fifth-order symmetric cubature rule [41] (i.e., the fifth-order unscented transform), the number of required sigma-points is \( 2m^2 + 1 \) and in the seventh-order rule it is \((4m^3 + 8m + 3)/3\) (see, e.g., [42]). Therefore, the total computational complexity per iteration is \( O(Tm^2) \) when using the fifth-order rule and \( O(Tm^3) \) when using the seventh-order rule.

B. Convergence Analysis

In this section, we study the local convergence of the FHDP method. It is already known that DDP converges quadratically to the unique minimizer \( u^n_{T-1} \) in well-defined conditions [14], [15], [16]. These results can be summarized as follows.

**Lemma 1 (DDP convergence):** Assume that \( \xi_k, k = 1, \ldots, T \) and \( x_k, k = 1, \ldots, T - 1 \) are three times continuously differentiable with respect to \( x_k \) and \( u_k \), and the second derivative of \( Q_k \) with respect to \( u_k \) is positive definite for all \( k \). Furthermore, assume that the iterates \( \{x_k(i), u_k(i), i = 1, T - 1\} \) produced by DDP are contained in a convex set \( D \), which also contains the minimizer \( \{x_k^n, u_k^n, i = 1, T - 1\} \). Then, the sequence of DDP iterates \( u_k(i) \) converges quadratically in the sense that where exist \( c > 0 \) such that

\[
\|u_k(i+1) - u_k^n\| \leq c\|u_k(i) - u_k^n\|^2. \tag{38}
\]

Our aim is now to prove the convergence of the proposed Fourier–Hermite dynamic programming (FHDP) method by constructing a modified model such that when we apply DDP on it, it exactly reproduces the FHDP result. For that purpose, let us first introduce the following lemma.

**Lemma 2 (Relationship of Taylor and Fourier–Hermite series):** Let us consider a scalar function \( g(y) \) and define the following Weierstrass type of transform:

\[
g(y) = \int g(z) N(z \mid y, \Sigma) dz. \tag{39}
\]

Let us also assume that \( g(y) \) is at least three times continuously differentiable, which can be ensured by, for example, \( \int \|z\|^{4n} |g(z)| N(z \mid y, \Sigma) dz < \infty \). Then, the Taylor series expansion of \( g(y) \) matches the Fourier–Hermite expansion of \( g(y) \) with respect to \( N(z \mid y, \Sigma) \) up to an additive constant.

**Proof:** When \( z \sim N(z \mid y, \Sigma) \), integration by parts gives

\[
E[g(z)H_2(\Lambda^{-1}(z - y))] = \Lambda^{-1} E[G_{zz}(z)] \Lambda
\]

where \( \Sigma = \Lambda \Lambda^{-1} \). Substituting to (25) then gives the following Fourier–Hermite series for \( g(y) \) with respect to \( N(z \mid y, \Sigma) \):

\[
g(y') \approx E[g(z)] + E[G_{zz}(z)] (y' - y) + \frac{1}{2} \text{tr} (E[G_{zz}(z)] \Sigma) \tag{41}
\]

where the expectations are over \( N(z \mid y, \Sigma) \).

For the Taylor series expansion of (39), we can change variables by \( \int g(z) N(z \mid y, \Sigma) dz = \int g(y + \xi) N(\xi \mid 0, \Sigma) d\xi \), which after differentiation under integral and changing back to \( z \) gives

\[
\hat{G}_y(y) = \int G_{zz}(z) N(z \mid y, \Sigma) dz \tag{42}
\]

In the notation of (41), we have \( \hat{g}(y) = E[g(z)], \hat{G}_y(y) = E[G_{zz}(z)], \) and \( \hat{G}_yy(y) = E[G_{zz}(z)] \), and hence the Taylor series expansion of \( \hat{g}(y') \) becomes

\[
\hat{g}(y') \approx E[g(z)] + E[G_{zz}(z)] (y' - y) + \frac{1}{2} (y' - y)^\top E[G_{zz}(z)] (y' - y) \tag{43}
\]
which is the same as (41) except for the last term which is constant in $\eta$. }

**Lemma 3 (Equivalent DDP model):** Consider a transformation of the problem (1) and (2), where we replace the end-condition with

$$\tilde{f}_T(x_T) = \int \tilde{f}_T(x'_T)N(x'_T \mid x_T, \Sigma_T) dx'_T$$

(44)

and the cost at time step $k$ by

$$\tilde{f}_k(x_k, u_k) = \frac{Q_k(x_k, u_k)}{\Sigma_k} - V_{k+1}(f_k(x_k, u_k))$$

(45)

where the transformed action-value function is defined by

$$\tilde{Q}_k(x_k, u_k) = \int \left[ f_k(x_k', u_k') + V_{k+1}(f_k(x_k', u_k')) \right] \times N([x_k', u_k'] \mid [x_k, u_k], \Sigma_k) du_k' dx_k'$$

(46)

Then, the iteration produced by applying DDP to the modified model exactly matches the iterations generated by FHDP.

**Proof:** Substituting the modified cost function (45) to (5) reduces the action value function to (46). Then by Lemma 2, the second-order Taylor series expansions of the action-value function taken around $x_T$ and $\hat{x}_k, \hat{u}_k$ match the Fourier–Hermite series expansions up to a constant, which only depends on the nominal trajectory. The maxima of the Taylor-series based action-value functions with respect to the $u_k$ also match the maxima obtained from the Fourier–Hermite series expansions (up to the constant). Therefore, the control laws are the same and as the forward passes are the same, the iterations produce exactly the same result.

**Theorem 1 (Quadratic Convergence of FHDP):** Assume that $f_k$ is three times continuously differentiable and the transformed problem defined in Lemma 3 has a unique solution $(x_1^*, u_1^*, \Sigma_1^*)$ within a convex set $D$. Further assume that the second derivatives of the transformed action-value function in (46) with respect to $u_k$ are positive definite and all the iterates produced by FHDP $(x_1^{(i)}, u_1^{(i)}, \Sigma_1^{(i)}, \Sigma_k^{(i)}) \in D$. Then, the sequence of iterates $\hat{u}_1^{(i)}$ produced by FHDP converges quadratically to the solution in the sense that where exist $\tilde{c} > 0$ such that

$$\|u_k^{(i+1)} - \hat{u}_k\| \leq \tilde{c} \|u_k^{(i)} - \hat{u}_k\|^2$$

(47)

**Proof:** By Lemma 3, FHDP can be seen as DDP, which finds the optimum of a transformed problem defined by (44), (45), and (46). The assumptions ensure that assumptions for Lemma 1 are satisfied which leads to the result.

**Remark 1:** Because the transformed model reduces to the original model when $\Sigma_T, \Sigma_k \to 0$, in this limit, the results of FHDP and DDP match.

**Remark 2:** In Theorem 1, we had to assume that the dynamics are three times differentiable to adapt the existing DDP convergence results to the current setting. However, as the Fourier–Hermite expansion is always formed for $Q_k$, the convergence result should apply even in the case that the dynamics are not three times differentiable as long as the transformed $\tilde{Q}_k$ is smooth, which it in very general conditions is.

### V. Experiments

To demonstrate the performance of our proposed method, we consider the classical pendulum and cart-pole models and compare to the classical DDP and the unscented DDP (UDP) in [18]. All the algorithms were implemented in MATLAB 2021b using its AD functionality. All experiments were carried on a CPU using AMD EPYC 7643 with 48 Cores and 2.3 GHz.

2The source code is available at https://github.com/EEA-sensors/FourierHermiteDynamicProgramming.git

---

Fig. 1. Reduction in cost for pendulum swing-up problem by using DDP, unscented DP (UDP), and sigma-point based FHDP (SPDP). Results of SPDP in (a) are with Gauss–Hermite rule with $p = 3$ (SPDP-GH) and different values of $\Sigma_T$ and $\Sigma_k$. In (b), the results of SPDP are with different integration rules: SPDP-GH is with the Gauss–Hermite ($p = 3$), third (SPDP-UT3), fifth (SPDP-UT5), and seventh (SPDP-UT7) order cubature/unscented rules.

**A. Pendulum Swinging Experiment**

First, we consider a pendulum swing-up problem, which was also used in [18]. The goal is to swing the pendulum from downward position ($\theta = 0$) to upward ($\theta = \pi$) by using an input torque $u$ as control. We define the state of the pendulum as $x = [\theta, \dot{\theta}]^T$, and use a quadratic cost function of the form

$$J = \frac{1}{2} \left( x_T - x_g \right)^\top W_T (x_T - x_g)$$

(48)

The parameters of the pendulum model and the cost function are the same as in [18]. We discretize the dynamics using a fourth-order Runge–Kutta method with a zero-order hold on $u$. The step size is set to 0.1 and $T = 50$. We set the initial and final states to be $x_1 = [0, 0]^T$ and $x_g = [\pi, 0]^T$, respectively.

Fig. 1(a) and (b) show the total cost of the trajectory as a function of the iteration number with DDP, UDP, and sigma-point based FHDP (SPDP) methods. As the aim is to compare the performance of DDP, UDP, and different variations of SPDP methods, in Fig. 1(a), we use SPDP with Gauss–Hermite quadrature rule of order $p = 3$ (we call this SPDP-GH) and set $\Sigma_k \in \{10^{-6} I, 10^{-3} I, 10^{-1} I \}$, where $I$ is the identity matrix and $\Sigma_T$ similarly. As can be seen from the figure, all the compared methods except for one SPDP converge to a very similar total cost. In the first few iterations, all methods have approximately similar total cost. In later iterations, however, SPDP-GH with the large covariance, say $\Sigma_T = 10^{-1} I$ and $\Sigma_k = 10^{-1} I$, is slower to reduce the cost and hence requires more iterations to converge. This is expected because a large value for the covariance corresponds to FH expansion, which averages the function over a large area around the nominal point. On the other hand, with a small covariance, say $\Sigma_T = 10^{-3} I$ and $\Sigma_k = 10^{-3} I$, the SPDP method coincides with DDP, which confirms the theoretical analysis of the method in Section IV-B. It can be seen that in this experiment both DDP and SPDP have better convergence speed than UDP. SPDP-GH with $\Sigma_T = 10^{-1} I$, and $\Sigma_k = 10^{-1} I$ has a slightly better cost reduction compared to DDP (see SPDP-GH ($10^{-6} I$) curve after 30 iterations).

Fig. 1(b) shows the performance of SPDP method with different sigma-point schemes. The schemes are Gauss–Hermite quadrature rule with $p = 3$, $\Sigma_T = 10^{-1} I$, and $\Sigma_k = 10^{-1} I$ (SPDP-GH), third (SPDP-UT3), fifth (SPDP-UT5), and seventh order (SPDP-UT7) unscented transforms, that is, spherical cubature rules [42]. We can see that the
third-order cubature/unscented rule is not sufficient for computing the integrals for Fourier–Hermite coefficients as discussed in Section III-C [see the curve of SPDP-UT3 in Fig. 1(b)]. The SPDP-GH, SPDP-UT5, and SPDP-UT7 methods converge with approximately similar number of iterations as DDP, and the performance is practically independent of the integration rule used.

Table I lists the average run times (in seconds) to compute backward and forward passes per iteration. As we can see, DDP requires more computational time due to computing the derivatives appearing in (8). UDP requires less time since the method avoids computing derivatives of \( f_k \). However, it requires computing the derivatives of \( l_k \) and backward propagation of sigma points. The computational speed of SPDP mainly depends on the number of sigma-points used in the integration rule. We also list the number of sigma-points that need to be evaluated for UDP and SPDP methods in Table I. The number of sigma points at terminal step \( T \) and at step \( k \) are denoted as \( m_T \) and \( m_k \), for \( k = 1, \ldots, T - 1 \). It is clear from this table that SPDP is faster than the other methods. The run times for SPDP-UT5 are the fastest among all the methods, because the number of evaluation points in SPDP-UT5 is the least of the SPDP methods.

### B. Cart-Pole Experiment

In this experiment, we consider a cart-pole balancing problem, where the aim is to balance the pole in upward position by applying an external force \( u \) to move the cart in the horizontal direction. The similar experiment was also performed in [18]. The cart with mass \( m_c \) is attached to a pole with mass \( m_p \) and length \( l \). We denote the state of this system as \( x = [v, \dot{v}, \theta, \dot{\theta}]^\top \), where \( v \) and \( \dot{v} \) are the position and velocity of the cart, respectively, and \( \theta \) and \( \dot{\theta} \) denote the angle and angular speed of the pole, respectively. We set the initial and final states to be \( x_1 = [0, 0, 0, 0]^\top \) and \( x_T = [0, \pi, 0, 0]^\top \), respectively. The differential equations of this cart-pole system can be found in [43]. We discretize the dynamics using fourth-order Runge–Kutta integration and zero-order hold for the control \( u \). We use a cost function of similar form as (48) and set the values \( m_p, m_c, l, g, W_T, W_e, \) and \( R \) to be the same as in [18]. The step size is set to be 0.1 and \( T = 50 \).

Similar to pendulum example, we investigate the performance of the methods in reducing total cost. The results of SPDP method with different covariances and different integration schemes are shown in Fig. 2(a) and (b). In this case, the covariance of SPDP affects the behavior more than in the pendulum experiment. For the first few iterations in Fig. 2(a), all methods have a fast cost reduction, fastest being SPDP with \( 10^{-1} I \) covariance. For the later iterations, the methods have different speeds of cost reduction. With larger covariances [see SPDP-GH (10^{-1} I) and SPDP-GH (10^{-4} I) in Fig. 2(a)], SPDP method is slower in reducing the total cost and does not reach convergence within the 100 iterations shown in the figure. With smaller covariances, SPDP has similar behavior as DDP. What is interesting in this figure is that SPDP-GH (10^{-4} I) has better cost reduction compared to DDP during intermediate iterations. The SPDP-GH (10^{-1} I) has the fastest cost reduction until the first six iterations. After that, there is no improvement. We also observed that the convergence of UDP method was the fastest among all the methods. In Fig. 2(b), we can see that the integration method has a slight effect on the performance, but the results of SPDPs with different integration rules are practically the same.

The average run times per iteration to compute the backward pass and forward passes are listed in Table II. As in the pendulum case, the SPDP methods are faster than the other methods, UT5-based method being fastest of them. However, the margin to the UDP method is now smaller as UDP requires a relatively smaller number of sigma points than SPDP methods.

### C. Quadcopter Experiment

Finally, we consider a multirotor uncrowed quadcopter, which has four rotors with six degrees of freedom [44]. The state of the system contains the 3-D coordinates, velocities, the orientation (roll, pitch, yaw), and the angular velocities. There are four control inputs consisting of the total thrust produced by four rotors and the input torques. The cost function is similar to (48) and fourth order Runge–Kutta method is used for discretization. Table III shows the results with different methods. DDP method is the slowest among all while SPDP-UT5 and SPDP-UT7 showed competitive results.
The SPDP-GH method is not feasible since the number of evaluation points increases exponentially as the number of dimensions increases.

VI. CONCLUSION AND DISCUSSION

In this article, we have proposed a Fourier–Hermite series based FHPD algorithm and its derivative-free implementation sigma-point dynamic programming (SPDP) that approximates the action-value function using Fourier–Hermite series and sigma points. This is in contrast to classical DDP, which is based on the use of Taylor series expansion. This new SPDP has the performance close or better than DDP algorithm and while it is computationally faster as the high order derivatives do not need to be evaluated. As shown by the experiments, it also can outperform another sigma-point-based DP method, unscented dynamic programming (UDP), and it is also computationally faster in the tested experiments. We have also proved the local second-order convergence of the proposed method.

Although in the second experiment, UDP outperformed the proposed method, in Fig. 2(a), we see that SPDP-GH with different $\Sigma_k, \Sigma_k$ produces different convergence behavior. Moreover, we notice a larger cost reduction in SPDP-GH ($10^{-1}$) than UDP until sixth iteration. This indicates that the covariance schedule of SPDP could be used to further improve the convergence speed of the method, and this is also confirmed by additional numerical experiments that we have done. However, we leave the further investigation of the covariance schedule as a future work.

ACKNOWLEDGMENT

Authors would like to thank Academy of Finland for funding.

REFERENCES