Symbolic Computation of Dynamics on Smooth Manifolds

Brian Bittner and Koushil Sreenath

Dept. of Mechanical Engineering, Carnegie Mellon University, Pittsburgh, USA.

Abstract. Computing dynamical equations of motion for systems that evolve on complex nonlinear manifolds in a coordinate-free manner is challenging. Current methods of deriving these dynamical models is only through cumbersome hand computations, requiring expert knowledge of the properties of the configuration manifold. Here, we present a symbolic toolbox that captures the dynamic properties of the configuration manifold, and procedurally generates the dynamical equations of motion for a great variety of systems that evolve on manifolds. Many automation techniques exist to compute equations of motion once the configuration manifold is parametrized in terms of local coordinates, however these methods produce equations of motion that are not globally valid and contain singularities. On the other hand, coordinate-free methods that explicitly employ variations on manifolds result in compact, singularity-free, and globally-valid equations of motion. Traditional symbolic tools are incapable of automating these symbolic computations, as they are predominantly based on scalar symbolic variables. Our approach uses Scala, a functional programming language, to capture scalar, vector, and matrix symbolic variables, as well as the associated mathematical rules and identities that define them. We present our algorithm, along with its performance, for computing the symbolic equations of motion for several systems whose dynamics evolve on manifolds such as $\mathbb{R}$, $\mathbb{R}^3$, S^2, $SO(3)$, and their product spaces.

1 Introduction

Computing dynamics of systems directly on nonlinear manifolds involves laborious manual computation and expert knowledge of the properties of the configuration space. The motivation for such computation lies in the end result of obtaining a compact, globally valid, and singularity free dynamical model. These traits enable the use of geometric controllers to obtain almost global stability properties. While such dynamical models for certain popular, modestly complex mechanical systems are readily available (through hand computations), there exists no procedural or automatic way to compute the symbolic dynamics on manifolds. The primary reason is that we do not have a way to capture non-scalar symbolic variables and perform non-scalar symbolic arithmetic. The contribution of this paper addresses this gap.

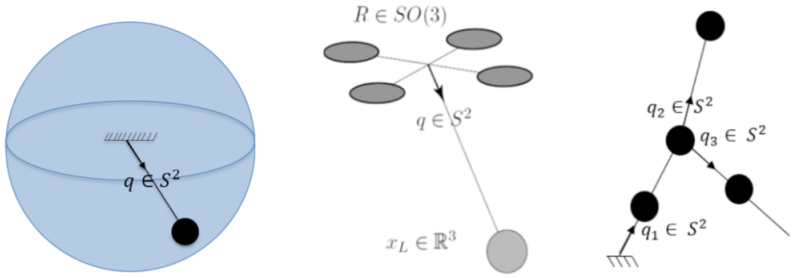


Fig. 1: The spherical pendulum, quadrotor with suspended load, and three-link walker systems that evolve on complex nonlinear manifolds. The dynamics for each are symbolically computed using the presented framework, see codebase at `https://gitlab.com/HybridRobotics/dynamics_on_manifolds`.

The paper can be summarized by the following: For systems that evolve on manifolds (see Figure 1), traditional ways to obtain dynamical models involve taking a local parametrization of the configuration space, such as Euler angles, and then solving for the dynamics. However, obtaining the dynamics on manifolds, without local parametrization, results in a dynamical model that is condensed to a compact expression, is globally valid, and is in a singularity free form. But, there exist no tools that can automate solving for the symbolic dynamics on manifolds. We provide a symbolic algebraic framework to capture geometric axioms and identities that govern the **scalar, vector, and matrix** elements of dynamical systems. An algorithm is shown to automate computation of the dynamics of simple mechanical systems on cartesian products of $\mathbb{C}^n$, $\mathbb{R}^n$, S^2, and $SO(3)$, as well as including computation of dynamics of a set of complex robotic systems.

The rest of the introduction places our proposed algorithm amongst prior work involving dynamics solvers and computer algebra systems, followed by an outline of the paper. With respect to dynamics solvers, Euler-Angle parametrization has long been the standard for system development. Introducing these local parametrizations of manifolds enables the use of well-established analytical methods and corresponding software algorithms for deriving the equations of motion. For instance, the Newton-Euler method is implemented in Neweul [16] and SD/FAST [14]. The Recursive Newton-Euler Algorithm has been used to efficiently compute dynamics of serial chain manipulators, [7,9]. The Lagrangian method has been implemented on various multibody systems [22]. Additionally, the Piogram Method [5], the Composite Rigid Body Algorithm [11], and the Articulated Body Algorithm [11], all provide efficient algorithmic approaches for solving the dynamical equations of long kinematic chains. Commercial systems such as ADAMS [24], SimMechanics, MapleSim, LMS Dads [27], etc., not only solve the equations of motion, but also offer efficient simulations. Symoro [15] allows for symbolic modeling and simulation for high dimensional kinematic tree-structures and closed kinematic chains. Although there are several efficient methods and corresponding software tools that automate the computation of

equations of motion, every single one of them is based on local parametrizations that employ *scalar* symbolic variables.

Automating symbolic computation on nonlinear manifolds requires treating vectors and matrices as single-entity symbolic variables with operations defined directly on them, which existing computing tools do not support. We achieve this by creating symbolic variables that represent either scalars, vectors, or matrices, with additional properties such as constants, unit vectors, symmetric matrices, etc. Moreover, using a functional programming approach, we enable a programmatic understanding of mathematical axioms and identities between these variables.

We looked at the field's best reputed computer algebra systems to encapsulate our dynamics framework. Commonly used Computer Algebra Systems include Mathematica [3] and Matlab's Symbolic Math Toolbox [2], (see [1] for full list of CASs). These only support scalar symbolic variables (vector and matrix symbolic variables exist only indirectly through scalars.) ATLAS 2 [10], an extension of Mathematica, enables high quality visualizing of n-dimensional manifolds, but is not applicable for our computation since it also requires local frame parametrizations. Our solution is to use Scala, a functional programming language with a vast set of object oriented programming capabilities [25]. This enables the design of strong expression classes and an intuitive functional programming environment for capturing mathematical identities. This coupled with existence of an extendable ScalaMathDSL [13] package made Scala preferable to the frameworks SymPy [30] and Axiom [8]. We extend prior work on scalar expressions in ScalaMathDSL [13] to handle vector and matrix classes, resulting in a powerful framework to implement the dynamics algorithm. Note that our modeling tool does not provide any type of numerical computation.

The rest of the paper is structured as follows: Section II, covers the mathematical background. Section III details the dynamics framework and walks through the algorithm to automate computation of the symbolic dynamics. Section IV tabulates results of applying the dynamics solver on a set of mechanical systems. Finally Sections V and VI provide a discussion and concluding remarks.

2 Mathematical Background

Many robotic systems evolve on nonlinear manifolds. However, we typically compute the dynamics of these systems through local parametrizations, such as Euler angles. Robotic systems are hindered by local parametrizations that induce singularities, model complexity, and unwanted features such as the unwinding phenomena [4] during control. The Lagrange-d'Alembert principle (see [20], [21], [18] for more details.), detailed in Section 2.1, allows the computation of dynamics of systems evolving on complex, nonlinear manifolds to arrive at singularity-free, globally valid, and compact equations of motion.

2.1 Lagrange d'Alambert Principle for Computing Dynamics

Let the degrees of freedom for a dynamical system exist on the configuration manifold Q. Then, the Lagrangian of the system, $\mathcal{L} : TQ \to \mathbb{R}$ is defined as $\mathcal{L} = \mathcal{T} - \mathcal{U}$, where $\mathcal{T} : TQ \to \mathbb{R}$ and $\mathcal{U} : Q \to \mathbb{R}$ are the kinetic and potential energies respectively. Here TQ is the tangent bundle of the configuration space. The virtual work $\mathcal{W} : Q \to \mathbb{R}$ captures energy input to the system via actuation. The dynamical equations of motion can be computed by minimization of the action integral S through the Least Action Principle [12], where the action integral is defined as:

$$S = \int_0^t \mathcal{L}\,\mathrm{d}t + \int_0^t \mathcal{W}\mathrm{d}t. \tag{1}$$

The minimization is done by setting the variation of the action integral to zero, i.e., $\delta S = 0$.

2.2 Computation on S^2 Manifold

To concretely illustrate the procedure, we carry out the following computational steps to derive the equations of motion of the spherical pendulum on S^2 [19]. (To observe the computation on $SO(3)$ the reader is directed to [6].)
Step 1: Supply the Lagrangian ($\mathcal{L}$) and virtual work ($\mathcal{W}$), providing the components of the action integral (S) for the spherical pendulum, as given below,

$$\mathcal{L} = \frac{1}{2}ml^2(\dot{q}\cdot\dot{q}) - mgl(q\cdot e_3), \qquad \mathcal{W} = q\cdot\tau,$$

$$S = \int_0^t \frac{1}{2}ml^2(\dot{q}\cdot\dot{q}) - mgl(q\cdot e_3)\ \mathrm{d}t + \int_0^t q\cdot\tau\ \mathrm{d}t.$$

Here unit vector $q \in S^2 \subset \mathbb{R}^3$ has the property $q\cdot q = 1$ and kinematic constraint $q\cdot\dot{q} = 0$. The constant scalars $(m, g, l) \in \mathbb{R}$ correspond to the mass of the pendulum, magnitude of acceleration due to gravity, and length of the pendulum respectively. Constant $-e_3 \in \mathbb{R}^3$ provides the direction of the gravitational field, while $\tau \in \mathbb{R}^3$ defines the torque via virtual work at the base of the pendulum. Generalized vector $\xi \in \mathbb{R}^3$ is orthogonal to the evolution of q such that the variation of q is given as, $\delta q = \xi \times q$.
Step 2: Take the variation of the action integral and set to zero:

$$\delta S = \int_0^t ml^2(\delta\dot{q}\cdot\dot{q}) - mgl(\delta q\cdot e_3)\,\mathrm{d}t + \int_0^t \delta q\cdot\tau\,\mathrm{d}t = 0.$$

Step 3: Replace variations δq and $\delta\dot{q}$ with the generalized vector ξ:

$$\int_0^t ml^2((\dot{\xi}\times q + \xi\times\dot{q})\cdot\dot{q}) - mgl(\xi\times q)\cdot e_3)\,\mathrm{d}t + \int_0^t (\xi\times q)\cdot\tau\,\mathrm{d}t = 0.$$

Step 4: Group equations with respect to $\dot{\xi}$ and ξ and apply simplifications (e.g.: $\dot{q}\times\dot{q} = 0$):

$$\int_0^t \dot{\xi}\cdot(ml^2 q\times\dot{q}) + \xi\cdot(ml^2\dot{q}\times\dot{q} - mglq\times e_3 + q\times\tau)\,\mathrm{d}t = 0.$$

Step 5: Perform integration by parts on $\dot{\xi}$:

$$\int_0^t -\xi \cdot (ml^2(\dot{q} \times \dot{q} + q \times \ddot{q})) + \xi \cdot (-mglq \times e_3 + q \times \tau)\,\mathrm{d}t = 0.$$

Step 6: Collect for ξ and simplify (e.g.: $\dot{q} \times \dot{q} = 0$):

$$\int_0^t \xi \cdot (-ml^2 q \times \ddot{q} - mglq \times e_3 + q \times \tau)\,\mathrm{d}t = 0.$$

Step 7: These equations of motion can be extracted to describe the dynamics:

$$\dot{q} = \omega \times q,$$
$$q \times (ml^2\ddot{q} + mgle_3) = q \times \tau,$$

where the first identity is a kinematic identity (derived from q's action on S^2) with ω being the angular velocity, and the second identity obtained from Step 6 by setting the term that is dotted with ξ as zero. This variation-based computation returns a set of compact, globally-valid equations of motion that are singularity-free. Having discussed the mathematical background, we will next describe the computational framework of our dynamics solver that automates this method.

3 Computational Design of Dynamics Solver

This section will provide a description of the data structures used to represent mathematical objects and cover the algorithmic techniques that operate on them. Critical computational steps in the algorithm are covered, followed by a user-level code snippet that implements the symbolic computation of dynamics on a spherical pendulum.

3.1 Class Hierarchies for Mathematical Expressions

In Section II we used symbolic scalars (m, l, g) and symbolic vectors (q, $\dot{q}$, e_3) to compute the dynamics for the spherical pendulum. These two expression types along with symbolic matrices make up the *base classes* used to implement the dynamics solver. The class hierarchy consists of each base class (ScalarExp, VectorExp, MatrixExp) and its children, which are comprised of a set of case classes. As we will see, *case classes* enable pattern matching to encode mathematical rules (e.g.: the chain rule for differentiation, scalar and vector triple product rules, etc.) Each case class can inherit *traits* to possess additional properties (e.g.: vectors having unit magnitude or matrices having skew symmetric properties.) The unit vector trait allows dynamics to be computed on the S^2 configuration space. The matrix traits constant, skew, and symmetric enable distinction of skew, skew symmetric, and rotation matrices. This enables direct and intelligent computation of dynamics on $SO(3)$. Each case class serves as a unique mathematical entity in the symbolic evaluator, see Figure 2 for the class hierarchy. Next we will discuss how these case classes interact to create a meaningful data structure for symbolic computation.

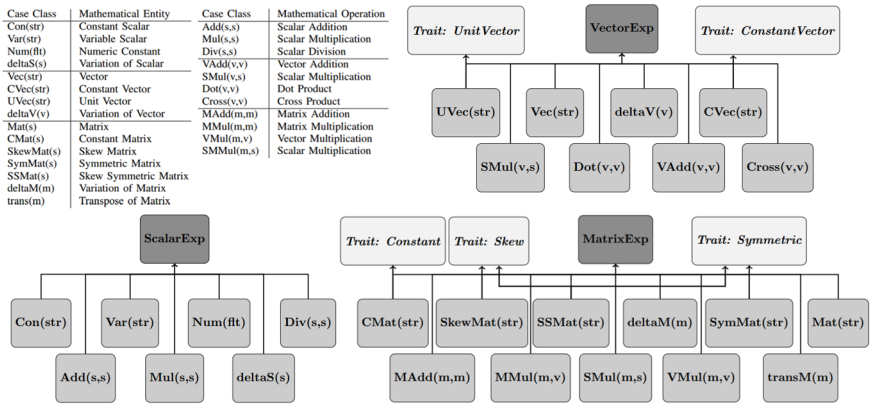


Fig. 2: All mathematical expressions inherit from a parent class corresponding to its identity as a scalar, vector, or matrix. Class traits are inherited to add additional properties to symbolic expressions.

3.2 Abstract Syntax Trees

Our mathematical expressions are stored as abstract syntax trees (ASTs). Mathematical operations, such as addition and multiplication, are represented as parent nodes which contain a specified number of leaves appropriate for their respective functionality. Leaves on the other hand represent symbolic scalars, vectors, and matrices. Mathematical algorithms are then written as functional programs with the ASTs serving as the input and output data structures. Figure 3 provides an AST of the Lagrangian for the spherical pendulum. As detailed next, ASTs are coupled with pattern matching functions to encode mathematical rules.

3.3 Embedding Mathematical Axioms

Writing the framework in Scala, a functional and object oriented programming language, we are able to succinctly embed important mathematical axioms governing the properties of the configuration manifolds. Because the elements of an AST are each designated a case class with inherited mathematical traits, this structure can be quickly parsed to enforce mathematical axioms and identities, which are used to simplify the dynamics and perform calculus operations.

3.4 Mathematical Operations

Each mathematical step in the computational pipeline, is achieved through specialized pattern matching functions. These functions perform a mathematical evaluation or restructuring of the AST. Pattern matching via the extensive class hierarchy for elements of manifold dynamics enables brief, yet powerful functions that are intuitive to write and understand.

To illustrate this, we will walk through the various functions implemented to execute each step of the variation-based method. Accompanied by illustrations of tree manipulation for a spherical pendulum, we discuss the procedural nature

of each step of the algorithm (steps enumerated 1-7). We will talk about trees of n nodes containing p parents (operands) and e literals (mathematical elements).

Tree Conditioning: For the steps below, the tree conditioning is defined as a full expansion of the expression, (i.e. $a * (b + c)$ is converted to $a * b + a * c$). Isolating the dynamical terms enable succinct, procedural code to be written for tree operations. Expanding the tree requires traversing the tree (covered next) and invokes 3 parent operations per expansion ($a * b + a * c$). A maximum of $e - 1$ expansions can take place during traversal. Table 1 provides a condensed layout of the computational cost for the algorithm.

Tree Traversal: Each algorithm requires traversal of the tree, from the top down. The general run time for this case is $O(n)$. If we balance the tree and parallelize the operations (allowed by the structured commutativity above), we could theoretically assert an $O(log(n))$ traversal time. The algorithm enables fast computation of the dynamics (see Results Tables 2 and 3), with most systems requiring computation time on the order of milliseconds.

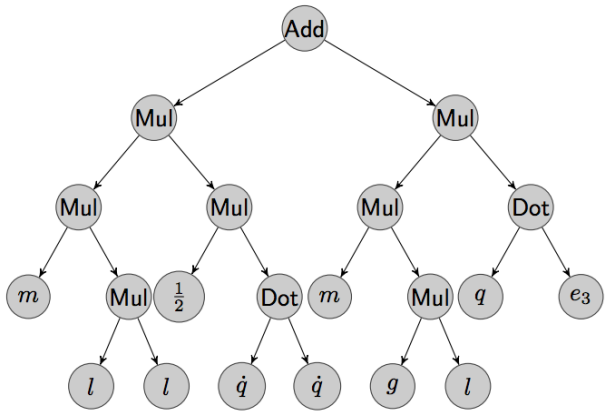


Fig. 3: **Step 1:** Abstract Syntax Tree representation of the Lagrangian of a spherical pendulum, given by $\mathcal{L} = \frac{1}{2}ml^2(\dot{q} \cdot \dot{q}) - mgl(q \cdot e_3)$. ASTs are the data structure used to represent all symbolic expressions.

Step 1 requires initialization of the Lagrangian AST. This involves supplying the information of the mechanical configuration and energy of the system, resulting in the input AST shown in Figure 3. The Lagrangian AST of e elements is constructed in $e - 1$ operations.

Step 2 takes the variation of the Lagrangian. To respect space constraints, we display variation of only the kinetic energy component of the Lagrangian AST in Figure 4a. Taking the variation of an element changes its string identifier and possibly its case class. For example, δq loses q's trait as a unit vector. This function also applies the chain rule, $\delta(u * v) = \delta u * v + u * \delta v$. The chain rule invokes 3 parent operations and can be applied $(e - 1)$ times. The variation function is run on each element once. Taking the time derivative has equivalent structure to this algorithm, but generates elements of different properties.

Step 3 shown in Figure 4b, applies constraints through substitution. The tree is searched for a node containing the expression candidate for

replacement. In this case, when $\delta\dot{q}$ is found, it is replaced with $\dot{\xi} \times q + \xi \times \dot{q}$. Substitution allows for the variations and kinematic constraints of the system to be enforced, along with simplifying expressions such as $\dot{q} \times \dot{q} = 0$. Each node is checked once. The tree is then **reconditioned** to fully expanded form.

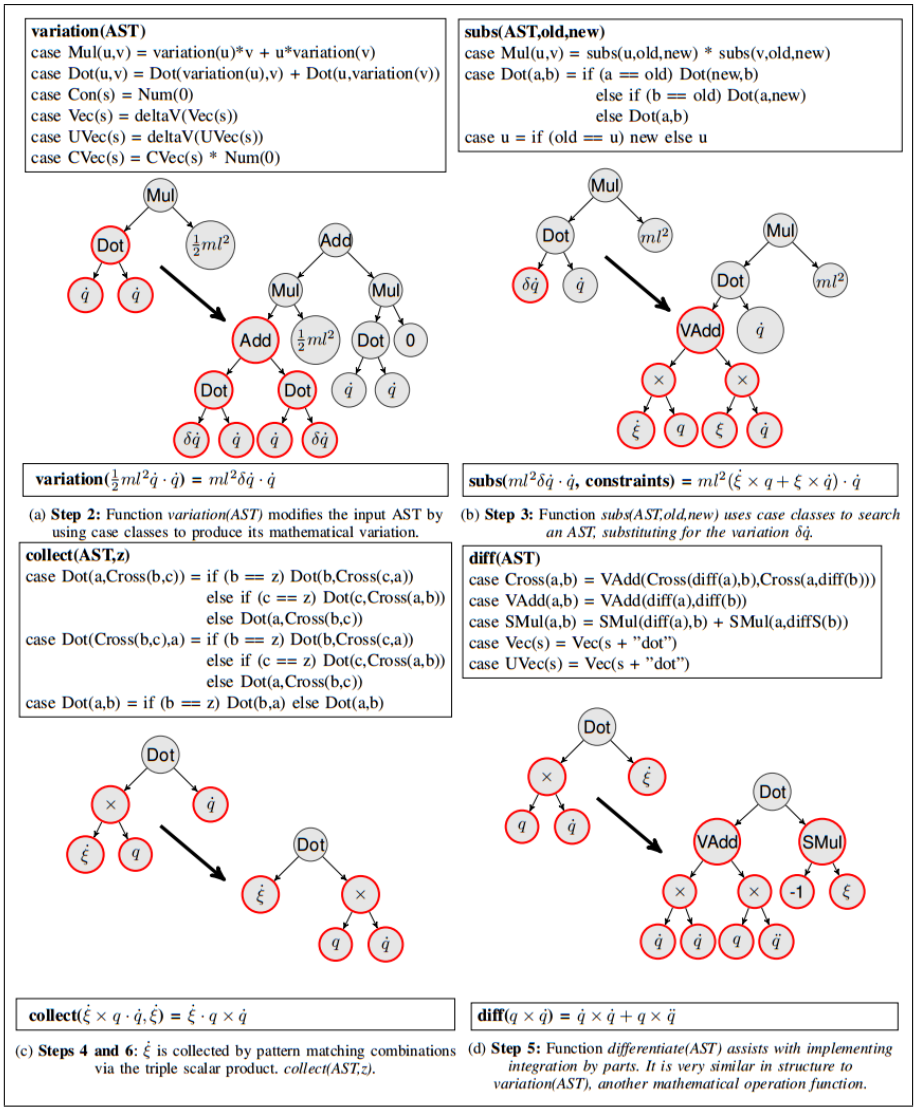
Step 4 collects an expression with respect to a vector or scalar. Figure 4c illustrates collecting for the generalized vector $\dot{\xi}$. For each isolated energy term, each element is checked for a match to the collected expression. A collected scalar is set as the leftmost multiplied element, allowed by commutativity. A collected vector is placed on the left hand side of the dot product by vector algebraic axioms such as the scalar triple product. Each collection requires 2 parent operations. Examples collecting for a include $b * (a * c) = a * (b * c)$ for scalars and $b \cdot (a \times c) = a \cdot (b \times c)$ for vectors.

Step 5 performs integration by parts on the dot product containing the collected generalized vectors from the previous step. In Figure 4d, the mathematical operation is enforced by negating and integrating $\dot{\xi}$ while differentiating the right hand side of the dot product. The result leaves the expression entirely in terms of generalized vector ξ. Since integration by parts requires two expressions, the negation, integration, and differentiation involved can operate on up to $e - 1$ elements. The tree is **reconditioned** again. This allows for the expression to be **collected with respect to ξ which is done in Step 6.** This allows for the right hand side of the dot product to be set to zero, resulting in the dynamical model obtained in **Step 7, where the dynamics can be extracted line by line,** one group of terms set to zero for each generalized vector. One parent operation is required to remove a term from the tree. This can occur e times.

Process	Traversal	Parent Operations	Literal Operations
Step 1:	-	$e-1$	e
Expand:	n	$3(e-1)$	-
Step 2:	n	$3(e-1)$	e
Step 3:	n	-	e
Expand:	n	$3(e-1)$	-
Step 4:	n	$2(e-1)$	-
Step 5:	n	$2(e-1)$	$3(e-1)$
Expand:	n	$3(e-1)$	-
Step 6:	n	$2(e-1)$	-
Step 7:	n	e	-
Run Time:	$O(n)$	$O(e)$	$O(e)$

Table 1: This table presents the run time of critical steps in the computation.

Linear run time convergence enables broad inspection of high degree of freedom dynamics, the laborious computational nature of which made model extraction previously intractable. The coupled simplicity and effectiveness of the algorithm is enabled by computing on coordinate free configuration spaces. There is no trigonometric book-keeping, which grows rapidly for systems described on S^1 and $SO(2)$.



(a) **Step 2:** Function *variation(AST)* modifies the input AST by using case classes to produce its mathematical variation.

(b) **Step 3:** Function *subs(AST,old,new) uses case classes to search an AST, substituting for the variation $\delta\dot{q}$.*

(c) **Steps 4 and 6**: $\dot{\xi}$ is collected by pattern matching combinations via the triple scalar product. *collect(AST,z).*

(d) **Step 5:** Function *differentiate(AST) assists with implementing integration by parts. It is very similar in structure to variation(AST), another mathematical operation function.*

Fig. 4: Illustration of various functions required for implementing the symbolic dynamics solver. Shown for each function are the input and output mathematical expressions, the corresponding input and output ASTs (with changes highlighted in red), and the corresponding code-snippets that invoke the tree manipulation.

3.5 Example with Spherical Pendulum

Listing 1 illustrates the user code for deriving the dynamics of a spherical pendulum. The user provides the Lagrangian and specifies the actuated variables of the system. These are input to the *computeDynamics* function (which implements the algorithm detailed in Section 3.4 to derive the dynamics for the system. Specification of configuration variables and manifold based constraints is not required, since the constraint information is encapsulated within mathematical type declaration of each variable. This simple example code will enable users to effortlessly set up problems for solving the dynamics of new robotic systems. Constraint manifolds are duduced from configuration variable types, i.e. $q \in S^2$ has kinematic constraint $\delta q = \xi \times q$. The codebase is available at `https://gitlab.com/HybridRobotics/dynamics_on_manifolds`.

```
object SphericalPendulum {
    def main() {
      // define constant scalars
      val g = Cons("g")  // gravitational constant
      val m = Cons("m")  // point mass
      val l = Cons("l")  // length at which point mass hangs

      // define vectors
      val e3  = CVec( "e3")  // orientation of gravity
      val  q  = UVec(  "q")  // point mass acts on S^2
      val  u  =  Vec(  "u")  // virtual work done on system

      // set configuration variables
      // (scalars,vectors,matrices)
      val configVars = Tuple3(List(),List(q),List())

       // define lagrangian
      val KE = Num(0.5) * m * l * l * (diffV(q) dot diffV(q))
      val PE = m * g * l * (q dot e3)
      val L = KE - PE

      // specify infinitesimal virtual work of system
      val infWork = Dot(deltaV(q),u)

      var eoms = computeDynamics(L, W, configVars)}
```

Listing 1: User code to compute dynamics for a spherical pendulum.

4 Results of Computing Symbolic Models

Having presented an overview of our computational framework for the symbolic dynamics solver, we next test this across many robotic systems with dynamics evolving on several different manifolds. Table 1 provides results on simple pendulum systems with dynamics on manifolds S^2 and $SO(3)$. Table 2, on the other

hand, introduces real robotic systems evolving on highly complex manifolds. The user supplies the Lagrangian, Virtual Work, and configuration variables. The table lists the runtime and equations produced for each mechanical system. Next, we briefly describe the systems tabulated in Table 1, for which our proposed symbolic algorithm computes the dynamics on manifolds.

4.1 Simple Mechanical Systems

A **Spherical Pendulum** [6] is the first system considered, its parametrization is described in section 2.2.
Configuration Manifold: S^2
Kinematic Constraints: $\dot{q} \cdot q = 0$
Variations: $\delta q = \xi \times q$, $\dot{\delta q} = \dot{\xi} \times q + \xi \times \dot{q}$, $\xi \in \mathbb{R}^3$
Energy Terms:

$$T = \frac{1}{2} m l^2 (\dot{q} \cdot \dot{q}), \qquad U = mgl(q \cdot e_3).$$

The **3D Pendulum** [28] provides a system which acts on the $SO(3)$ manifold. The pendulum is represented as a rigid body of mass m and symmetric inertia matrix $J \in \mathbb{R}^{3\times3}$. It is pivoted at the point from which vector $\rho \in \mathbb{R}^3$ extends toward the center of mass. The orientation of the 3D pendulum is specified by rotation matrix $R \in SO(3)$ with $\Omega, M \in \mathbb{R}^3$ being the angular velocity and moment input of the 3D pendulum respectively. The generalized vector $\eta \in \mathbb{R}^3$ is utilized to describe the variation of R on $SO(3)$.
Configuration Manifold: $SO(3)$
Kinematic Constraints: $\dot{R} = R\hat{\Omega}$
Variations: $\delta R = R\hat{\eta}$, $\delta\Omega = \dot{\hat{\Omega}}\eta + \dot{\eta}$, $\eta \in \mathbb{R}^3$
Energy Terms:

$$T = \frac{1}{2} \Omega \cdot J\Omega, \qquad U = mgR\rho \cdot e_3.$$

A **Double Spherical Pendulum** [20] is an extension of the singular Spherical Pendulum. A second pendulum with point mass m_2 is pivoted from the end of the first pendulum containing point mass m_1. Each pendulum has a respective length l_i from its pivot to point mass. The unit vector $q_i \in S^2$ points along each l_i with its origin at the respective pivots. The torque at the base of each pendulum is given by $\tau_i \in \mathbb{R}^3$.
Configuration Manifold: $S^2 \times S^2$
Kinematic Constraints: $\dot{q}_i \cdot q_i = 0$
Variations: $\delta q_i = \xi_i \times q_i$, $\dot{\delta q_i} = \dot{\xi}_i \times q_i + \xi_i \times \dot{q}_i$, $\xi_i \in \mathbb{R}^3$
Energy Terms:

$$T = \frac{1}{2}(m_1 + m_2) l_1^2 \dot{q_1} \cdot \dot{q_1} + m_1 l_1 l_2 \dot{q_1} \cdot \dot{q_2} + \frac{1}{2} m_2 l_2^2 \dot{q_2} \cdot \dot{q_2}$$
$$U = (m_1 + m_2) g l_1 q_1 \cdot e_3 + m_2 g l_2 q_2 \cdot e_3$$

Similarly, the **Double 3D Pendulum** [17] is an extension of the Singular 3D Pendulum. The rigid body of mass m_2 is pivoted from the point that vector $\lambda \in \mathbb{R}^3$ points to. The rigid bodies' orientations at point masses m_i are specified

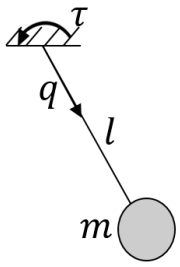
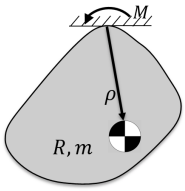
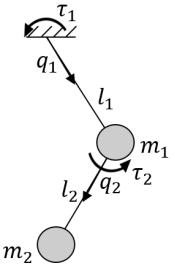
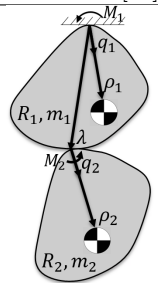
System	Config. Manifold	DOF	Time to Compute	Equations of Motion Produced by our Symbolic Dynamics Solver
 Spherical Pendulum [6]	S^2	2	0.019 sec.	$\dot{q} = \omega \times q$ $q \times (ml^2\ddot{q} + mgle_3) = q \times \tau$
 3D Pendulum [28]	$SO(3)$	3	0.015 sec.	$\dot{R} = R\hat{\Omega}$ $J\dot{\Omega} + \Omega \times J\Omega = mg\rho \times R^T e_3 + M$
 Double Spherical Pendulum [19]	$S^2 \times S^2$	4	0.032 sec.	$\dot{q}_1 = \omega_1 \times q_1$ $q_1 \times ((m_1+m_2)l_1^2\ddot{q}_1 + m_2 l_1 l_2 \ddot{q}_2 + (m_1+m_2)gl_1e_3) = q_1 \times \tau_1$ $\dot{q}_2 = \omega_2 \times q_2$ $q_2 \times (m_2 l_1 l_2 \ddot{q}_1 + m_2 l_2^2 \ddot{q}_2 + m_2 g l_2 e_3) = q_2 \times \tau_2$
 Double 3D Pendulum [17]	$SO(3) \times SO(3)$	6	0.033 sec.	$(J_1 - m_1\hat{\lambda}^2 - m_2\hat{\rho}_2^2)\dot{\Omega}_1 + m_2\hat{\rho}_1 R_1^T R_2 \hat{\rho}_2 \dot{\Omega}_2 + \hat{\Omega}_1(J_1 - m_1\hat{\lambda}^2 - m_2\hat{\rho}_1^2)\Omega_1 + m_1 g\hat{\lambda}R_1^T e_3 + m_2 g\hat{l}_1 R_1^T e_3 + m_2\hat{\rho}_1 R_1^T R_2 \hat{\Omega}_2^2 \lambda = M_1$ $(J_2 - m_2\hat{\rho}_2^2)\dot{\Omega}_2 - m_2\hat{\rho}_2 R_2^T R_1 \hat{\rho}_1 \dot{\Omega}_1 + \hat{\Omega}_2(J_2 - m_2\hat{\rho}_2^2)\Omega_2 + m_2 g\hat{\rho}_2 R_2^T e_3 + m_2\hat{\rho}_2 R_2^T R_1 \hat{\Omega}_1^2 \rho_1 = M_2$

Table 2: The equations of motion were computed on these simple systems using the proposed symbolic computation tool with the kinematic constraints, variations, and Lagrangian (T-U) provided for each system.

by rotation matrices R_i.
Configuration Manifold: $SO(3) \times SO(3)$
Kinematic Constraints: $\dot{R} = R\hat{\Omega}$
Variations: $\delta R_i = R_i \hat{\eta}_i$, $\delta \Omega_i = \dot{\hat{\Omega}}_i \eta_i + \hat{\Omega}_i \dot{\eta}_i$, $\eta_i \in \mathbb{R}^3$
Energy Terms:

$$T = \frac{1}{2}\Omega_1 \cdot J_1 \Omega_1 + \frac{1}{2}\Omega_2 \cdot J_2 \Omega_2 + \frac{1}{2} m_2 \dot{x_2} \cdot \dot{x_2}$$
$$U = m_1 g R_1 \rho_1 \cdot e_3 + m_2 g R_2 \rho_2 \cdot e_3$$

4.2 Robotic Systems

Having seen the algorithm work for simple mechanical systems, we next consider robotic systems with more complex configuration spaces, shown in Table 3.

The **Three Link Walker** [23] represents an idealized mechanical configuration for bipedal walkers. Masses m, m_H, and m_T, are dynamically actuated by torques τ_2 and τ_3. Unit vectors q_1, q_2, and q_3 specify the motion of the three links. The **Stance Dynamics** are as shown below. The **Flight Dynamics** exist on $S^2 \times S^2 \times S^2 \times \mathbb{R}^3$, and account for the walkers evolution in Euclidean Space.
Configuration Manifold: $S^2 \times S^2 \times S^2$
Kinematic Constraints: $\dot{q_i} \cdot q_i = 0$
Variations: $\delta q_i = \xi_i \times q_i$, $\delta \dot{q_i} = \dot{\xi}_i \times q_i + \xi_i \times \dot{q_i}$
Energy Terms:

$$T = (\frac{5m}{8} + \frac{m_H}{2} + \frac{m_T}{2}) l^2 \dot{q_1} \cdot \dot{q_1} + l^2 (\frac{m}{2}\dot{q_1} + \frac{m}{8}\dot{q_2}) \cdot \dot{q_2} + \frac{m_T}{2}(lL\dot{q_1} + L^2 \dot{q_2}) \cdot \dot{q_3}$$
$$U = (\frac{3m}{2} gl + m_H l + m_T gl) q_1 \cdot e_3 \frac{m}{2} gl q_2 \cdot e_3 + m_T gL q_3 \cdot e_3$$

The **Quadrotor with Tethered Load** [29] and **Reaction Mass Pendulum** [26] can be referred to at the respective sources for information on configuration variables and composition of the Lagrangian.

Manifolds $\mathbb{C}^n$ and $\mathbb{R}^n$ are directly computed by the same steps shown in Section III, with the exception of Step 3, which introduces constraint manifolds for S^2 and $SO(3)$. This step is not required for variables that act on $\mathbb{C}^n$ and $\mathbb{R}^n$. The computational results for each system were validated by a published derivation and hand calculation. These results establish the validity and efficiency of computing dynamics on complex manifolds using the symbolic evaluator.

5 Discussion

It should be noted that manually computing and validating the dynamics for several of the systems in Table 3 would have taken a novice one to several days. Our proposed method of automating the computation reduces this to fractions of seconds. This will enable a broader study of novel robotic systems whose dynamics evolve on complex manifolds, systems that were previously not broadly

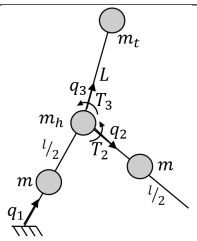
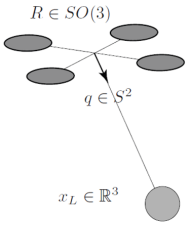
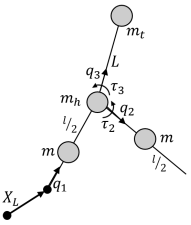
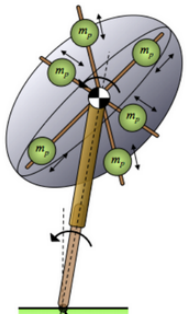
System	Config. Manifold	DOF	Time to Compute	Equations of Motion Produced by our Symbolic Dynamics Solver
 3-Link Walker Stance Dynamics [23]	$S^2 \times S^2 \times S^2$	6	0.032 sec.	$\dot{q}_1 = \omega_1 \times q_1$ $q_1 \times ((\frac{5}{4}m + m_H + m_T)l^2\ddot{q}_1 + \frac{1}{2}ml^2\ddot{q}_2 + m_T lL\ddot{q}_3 + (\frac{3}{2}m + m_H + m_T)gle_3) = 0$ $\dot{q}_2 = \omega_2 \times q_2$ $q_2 \times (\frac{1}{2}ml^2\ddot{q}_1 + \frac{1}{4}ml^2\ddot{q}_2 + \frac{1}{2}mgle_3 = q_2 \times \tau_2$ $\dot{q}_3 = \omega_3 \times q_3$ $q_3 \times (m_T lL\ddot{q}_1 + m_T L^2\ddot{q}_3 + m_T gLe_3 = q_3 \times \tau_3$
 Quadrotor with Suspended Load [29]	$SO(3) \times S^2 \times \mathbb{R}^3$	8	0.024 sec.	$\dot{x_L} = v_L$ $(m_Q + m_L)\dot{V}_L + m_Q l(\dot{q} \cdot \dot{q})q + (m_Q + m_L)ge_3 = fRe_3 + p \times (p \times fRe_3)$ $\dot{q} = \omega \times q$ $q \times (ml^2\ddot{q} + mgle_3) = q \times T$ $\dot{R} = R\hat{\Omega}$ $J\dot{\Omega} + \Omega \times J\Omega = M$
 3 Link Walker Flight Dynamics [23]	$S^2 \times S^2 \times S^2 \times \mathbb{R}^3$	10	0.064 sec.	$\dot{q}_1 = \omega_1 \times q_1$ $q_1 \times ((\frac{5}{4}m + m_h + m_t)l^2\ddot{q}_1 + \frac{1}{2}ml^2\ddot{q}_2 + m_t lL\ddot{q}_3 + (\frac{7}{4}m + m_h + 2m_t)l\ddot{x} + (\frac{3}{2}m + m_h + m_t)lge_3) = 0$ $\dot{q}_2 = \omega_2 \times q_2$ $q_2 \times (\frac{1}{2}ml^2\ddot{q}_1 + \frac{1}{4}ml^2\ddot{q}_2 + \frac{3}{4}ml\ddot{x} + \frac{1}{2}mlge_3) = q_2 \times \tau_2$ $\dot{q}_3 = \omega_3 \times q_3$ $q_3 \times (m_t lL\ddot{q}_1 + m_t L^2\ddot{q}_3 + 2m_t L\ddot{x} + m_t Lge_3) = q_3 \times \tau_3$ $(\frac{7}{4}lm + lmh + 2lmt)\ddot{q}_1 + (\frac{3}{4}ml + \frac{1}{2}m_t L)\ddot{q}_2 + \frac{3}{2}m_t L\ddot{q}_3 + (\frac{5}{2}m + mh + 4m_t)\ddot{x} + (2m + 1mh + 2mt)ge_3 = 0$
 Reaction Mass Pendulum [26]	$\mathbb{C} \times SO(3) \times SO(3) \times S$	11	0.073 sec.	$m\ddot{\rho} = -m\rho\Omega_L^T\hat{e_3}^2\Omega_L + mge_3^T R_L^T e_3$ $\dot{R}_L = R_L\hat{\Omega}_L$ $J_L\dot{\Omega}_L = -\Omega_L \times J_L\Omega_L + 2m\rho\dot{\rho}\hat{e_3}^2\Omega_L + mg\rho\hat{e_3}R_L^T e_3 + \tau_L - R_L^T R_P\tau_D$ $\dot{R}_P = R_P\hat{\Omega}_P$ $J_P\dot{\Omega}_P = -\Omega_P \times J_P\Omega_P + 4\sum m_p s_i\dot{s}_i\hat{e}_i^2\Omega_P + \tau_D$ $2m_P\ddot{s}_i = -2m_P s_i\Omega_P^T\hat{e}_i^2\Omega_P + U_i$

Table 3: The equations of motion were computed on these robotic systems using the proposed symbolic computation tool with the kinematic constraints, variations, and Lagrangian (T-U) provided for each system.

approachable by the robotics community. The compact models will also bring new insight into the dynamics of existing robotic systems, enabling novel control designs for achieving highly dynamical maneuvers. A limitation of this method is that it cannot evaluate systems with nonholonomic constraints. Dynamics can only be computed on Cartesian products of $\mathbb{C}^n$, $\mathbb{R}^n$, S^2, and $SO(3)$.

6 Conclusion

We have presented an algorithm which automates the computation of dynamical equations for systems evolving on manifolds. By utilizing pattern matching within the Scala framework, we are able to capture the geometric axioms and identities of scalar, vector, and matrix elements of a dynamical system. Using the framework we implement a generalized algorithm that works across a wide variety of manifolds. The time efficient computation allows for the software to provide near-instantaneous output of dynamical equations. The dynamics generated are compact, globally-valid, and free of singularities, as they are described directly on the configuration manifold. This tool is released publicly and will enable broader inspection of systems that act on complex, nonlinear manifolds.

7 Acknowledgements

B. Bittner would like to thank Carnegie Mellon's URO, who supported him with a Summer Undergraduate Research Fellowship. K. Sreenath would like to thank R. Ravindran for discussions that led to the use of Scala. This work is supported by NSF grants IIS-1464337 and CMMI-1538869.

References

1. List of computer algebra systems. http://en.wikipedia.org/wiki/List_of_computer_algebra_systems.
2. *Symbolic Math Toolbox*. Mathworks Inc, 1993.
3. *Mathematica Version 10.0*. Wolfram Research Inc., 2014.
4. S. P. Bhat and D. S. Bernstein, "A topological obstruction to continuous global stabilization of rotational motion and the unwinding phenomenon," *Systems & Control Letters*, vol. 39, no. 1, pp. 63–70, 2000.
5. P.-y. Cheng, C.-i. Weng, and C.-k. Chen, "Symbolic derivation of dynamic equations of motion for robot manipulators using piogram symbolic method," *IEEE Journal of Robotics and Automation*, vol. 4, no. 6, pp. 599–609, 1988.
6. L. Consolini and M. Tosques, "On the exact tracking of the spherical inverted pendulum via an homotopy method," *Systems & Control Letters*, vol. 58, no. 1, pp. 1–6, 2009.
7. P. I. Corke, "An automated symbolic and numeric procedure for manipulator rigid-body dynamic significance analysis and simplification," in *IEEE Conf. on Robotics and Automation*, 1996, pp. 1018–1023.
8. T. Daly, D. V. Chudnovsky, and G. V. Chudnovsky, *Axiom The 30 Year Horizon*. The Axiom Foundation, 2007.

9. E. Dean-leon, S. Nair, and A. Knoll, "User friendly matlab-toolbox for symbolic robot dynamic modeling used for control design," in *IEEE Conf. on Robotics and Biomimetics*, 2012, pp. 2181–2188.
10. I. DigiArea, *Atlas 2 for Mathematica*, 2012. [Online]. Available: http://www.digi-area.com/Mathematica/atlas/guide/Atlas.php
11. R. Featherstone and D. Orin, "Robot dynamics: equations and algorithms," in *IEEE Conf. on Robotics and Automation*, 2000, pp. 826–834.
12. R. P. Feynman, "The principle of least action in quantum mechanics," Ph.D. dissertation, Princeton University Princeton, New Jersey, 1942.
13. T. Flaherty, "Scala Math DSL," https://github.com/axiom6/ScalaMathDSL, 2013.
14. M. G. Hollars, D. E. Rosenthal, and M. A. Sherman, *SD/FAST users manual*. Symbolic Dynamics Inc, 1991.
15. W. Khalil, F. Bennis, C. Chevallereau, and J. Kleinfinger, "Symoro: A software package for the symbolic modelling of robots," in *Proc. of the 20th ISIR*, 1989.
16. E. Kreuzer and W. Schiehlen, "Neweul software for the generation of symbolical equations of motion," in *Multibody Systems Handbook*. Springer, 1990.
17. T. Lee, "Computational geometric mechanics and control of rigid bodies," Ph.D. dissertation, University of Michigan, Ann Arbor, 2008.
18. T. Lee, M. Leok, and N. H. McClamroch, "Lagrangian mechanics and variational integrators on two-spheres," *International Journal for Numerical Methods in Engineering*, vol. 79, no. 9, pp. 1147–1174, 2009.
19. ——, "Lagrangian mechanics and variational integrators on two-spheres," *J. for Numerical Methods in Engineering*, vol. 79, pp. 1147–1174, 2009.
20. ——, "Stable manifolds of saddle equilibria for pendulum dynamics on S2 and SO(3)," in *IEEE Conf. on Decision and Control*, 2011, pp. 3915–3921.
21. ——, "Dynamics and control of a chain pendulum on a cart," in *IEEE Conf. on Decision and Control*, 2012, pp. 2502–2508.
22. R. Lot and M. D. A. Lio, "A symbolic approach for automatic generation of the equations of motion of multibody systems," *Multibody System Dynamics*, vol. 12, pp. 147–172, 2004.
23. P. X. Miranda, L. Hera, A. S. Shiriaev, B. Freidovich, S. Member, U. Mettin, and S. V. Gusev, "Stable walking gaits for a three-link planar biped robot with one actuator," *IEEE Trans. on Robotics*, vol. 29, no. 3, pp. 589–601, 2013.
24. G. Nonlinearity, "Adams 2014," pp. 1–5, 2014.
25. M. Odersky and Al., "An Overview of the Scala Programming Language," EPFL, Lausanne, Switzerland, Tech. Rep. IC/2004/64, 2004.
26. A. K. Sanyal and A. Goswami, "Dynamics and balance control of the reaction mass pendulum: A three-dimensional multibody pendulum with variable body inertia," *J. Dyn. Sys., Meas., Control*, vol. 136, no. 2, p. 021002, Nov. 2013.
27. D. Services, "LMS Virtual . Lab The Unified Environment for Functional Performance Engineering," pp. 1–16, 2007.
28. J. Shen, A. K. Sanyal, N. A. Chaturvedi, D. Bernstein, and H. McClamroch, "Dynamics and control of a 3d pendulum," in *IEEE Conf. on Decision and Control*, 2004, pp. 323–328.
29. K. Sreenath, T. Lee, and V. Kumar, "Geometric control and differential flatness of a quadrotor UAV with a cable-suspended load," in *IEEE Conf. on Decision and Control*, 2013, pp. 2269–2274.
30. SymPy Development Team, *SymPy: Python library for symbolic mathematics*, 2014. [Online]. Available: http://www.sympy.org