AlgebraicMoments

AlgebraicMoments is a package for automatically generating code to propagate statistical moments through polynomial expressions (constraints or dynamics).

Background

Statisticial moments are a useful way of characterizing distributions of random variables. Motivated by the problem of chance-constrained motion planning for potentially non-Gaussian systems with nonlinear constraints and dynamics, recent works leverage statistical moments to establish upper bounds on chance-constraints. Doing so, however, often requires the user to either propagate statistical moments through dynamics or compute moments of polynomials applied to random vectors. It turns out that, when working with polynomial dynamics and constraints, it is possible to algorithmically derive closed form expressions for the necessary moments.

Quick Start

At the repo root, enter:

```
pip3 install -e .
```

You should then be able to `import algebraic_moments`.

Github and Markdown Math

Unfortunately, Github currently does not currently render mathematical expressions in Markdown, so the user is advised to render this README.md themselves (for example, using VSCode + the mdmath extension).

Moment Expressions

Letting $\mathbf{w}$ denote a random vector, $\mathbf{y}$ denote deterministic variables, and g denote a polynomial function. The moment expressions capability allows the user to express:

$$\mathbb{E}[g(\mathbf{y}, \mathbf{w})^n]$$

in terms of $\mathbf{y}$ and moments of $\mathbf{w}$ in closed form. In the case that g is a vector-valued function, AlgebraicMoments can also handle moments of the following form where $n_i\in\mathbb{N}$:

$$\mathbb{E}\left[\prod_{i=1}^n g(\mathbf{y}, \mathbf{w})^{n_i}\right]$$

To simplify notation, we express moments of the above form with multi-index notation. Letting $\alpha\in\mathbb{N}^n$ denote a multi-index and α_i denote the i_{th} element of α, multi-index notation is defined as:

$$\mathbb{E}[g(\mathbf{y}, \mathbf{w})^\alpha] := \mathbb{E}\left[\prod_{i=1}^n g(\mathbf{y}, \mathbf{w})^{\alpha_i}\right]$$

Using AlgebraicMoments to Derive Moment Expressions

Suppose $\mathbf{x} = [x, y]$ is a random vector with x, y dependent and c is some deterministic variable. Now suppose we are interested in expressing the following moments in terms of moments of x, y:

$$g_1 = \mathbb{E}[(cxy^2 + y)^2]\ g_2 = \mathbb{E}[(x^2y + cy^2)^3]$$

We can express this problem using AlgebraicMoments as simply as:

```
# Specify elements of the random vector and dependencies.
x = RandomVariable("x")
y = RandomVariable("y")
dependencies = [(x, y)] # Specify pairwise dependencies between variables
random_vector = RandomVector([x, y], dependencies)

# Specify the deterministic variables.
c = DeterministicVariable("c")
deterministic_variables = [c]

# Specify the moments we care about as a dictionary, with the keys being
the name we want to give the moment and the values being the expression in
terms of our random and deterministic variables.
expressions = {"g1" : (c * x*y**2 + y)**2, "g2" : (y*x**2 + c*y**2)**3}
```

And then generating code for our moment expressions is as simple as:

```
# Generate an instance of "MomentExpressions" and print
moment_expressions = generate_moment_expressions(expressions,
random_vector, deterministic_variables)
moment_expressions.print("matlab")
```

The output to the terminal from executing this code is:

```
% Parse required inputs.
xPow2 = input_moments.xPow2;
yPow4 = input_moments.yPow4;
xPow1 = input_moments.xPow1;
yPow3 = input_moments.yPow3;
yPow2 = input_moments.yPow2;
xPow6 = input_moments.xPow6;
xPow4 = input_moments.xPow4;
yPow5 = input_moments.yPow5;
yPow6 = input_moments.yPow6;
c = input_deterministic.c;

% Moment expressions.
```

```
g1 = c.^2.*xPow2.*yPow4 + 2*c.*xPow1.*yPow3 + yPow2;
g2 = c.^3.*yPow6 + 3*c.^2.*xPow2.*yPow5 + 3*c.*xPow4.*yPow4 + xPow6.*yPow3;
```

Concentration Inequalities

For an uncertain ϵ chance-constraint expressed as:

$$\mathbb{P}(g(\mathbf{y}, \mathbf{w})\leq 0)\leq \epsilon$$

Given the mean and variance of $g(\mathbf{y}, \mathbf{w})$, we can upper bound the above probability using concentration inequalities. This package provides the ability to generate code for the following concentration inequalities:

- Cantelli
- Vysochanskij-Petunin (VP)
- Gauss

Going back to the example for MomentExpressions, suppose we are interested in bounding:

$$\mathbb{P}((cxy^2 + y)^2\leq 0)$$

We can generate the corresponding code with:

```
concentration_inequality = generate_concentration_inequality((c * x*y**2 +
y)**2, random_vec, deterministic_vars, "cantelli")
concentration_inequality.print("python")
```

Dynamical Systems (WORK IN PROGRESS)

Let $\mathbf{x}_t$ denote the state, $\mathbf{u}_t$ denote the control, and $\mathbf{w}_t$ denote a random vector that is independent of $\mathbf{x}_t$. Suppose we have a polynomial system:

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_t)$$

In many cases, we want to be able to determine statistical moments of $\mathbf{x}_t$ for some finite horizion (perhaps we want the mean vector and covariance matrix). In a certain sense, we are trying to determine the dynamics of a moment state. In this package, we introduce two methods for doing so 1) moment shooting and 2) moment state dynamical systems.

Moment Shooting

Moment shooting is analogous to direct shooting in trajectory optimization, we essentially express the statistical moments of $\mathbf{x}_t$ in close form in terms of moments of the initial state distribution $\mathbf{x}_0$, the control inputs $\mathbf{u}_t$, and moments of the disturbance vector $\mathbf{w}_t$.

Moment State Dynamical System (MSDS)

The method of moment state dynamical systems (MSDS) involves finding a closed form system of equations that expresses the dynamics of a set of moments we care about. Instead of expressing the dynamics in terms of the state vector $\mathbf{x}_t$, a MSDS expresses the dynamics of a set of moments of $\mathbf{x}_t$. With this package, the user can specify a system and a set of moments they care about. Before going any further, we need to introduce some notation. Recall multi-index notation from the section on Moment Expressions; note how there is a one-to-one mapping between statistical moments of $\mathbf{x}_t$ and multi-indices in $\mathbb{N}^n$. Thus, we can represent moments with multi-indices, and we will call sets of multi-indices "moment bases" and the corresponding set of moments "moment states". Letting $\mathcal{A}$ denote a moment basis, the moment state will be denoted by:

$$\mathbf{x}_t[\mathcal{A}]:={\mathbb{E}[\mathbf{x}_t^\alpha] : \alpha\in\mathcal{A}}$$

As a simple example, consider $\mathbf{x} = [x, y]$ with the moment basis $\mathcal{A} = {(1,0), (0, 1), (2, 0)}$. The moment state would then be:

$$\mathbf{x}[\mathcal{A}] = {x, y, x^2}$$

The idea is that the user can specify a dynamical system and a moment basis (or state) that they care about and then an algorithm called TreeRing (named as such because it is essentially tree search over the ring of polynomials) will search for a higher dimensional moment basis $\mathcal{A}$ and the corresponding set of dynamics equations h such that:

$$\mathbf{x}_{t+1}[\mathcal{A}] = h(\mathbf{x}_t[\mathcal{A}],\mathbf{w}_t[\mathcal{B}], \mathbf{u}_t)$$

Above $\mathcal{B}$ is a moment basis for the disturbance vector and $\mathbf{w}_t[\mathcal{B}]$ denotes a set of moments of the disturbance vector. This is a powerful idea because we can simulate the statistical moments of state in the future in a manner that scales linearly w.r.t. the number of time steps.

Conventions

Code Generation Programming Languages

The currently supported languages are:

```
- "matlab"
- "python"
- "octave"
```

Moment `string_rep`

- str(variable.string_rep) + str(variable.power)

Variable Name Rules

- No numbers
- No subscripts
- Only continuous letters