**Abstract**

The following presents a subset of the `spot/nlid` nonlinear identification utilities. The vast majority of SPOT is the product of Alexandre Megretski. These tools leverage SPOT and semidefinite solvers to optimize models based on Robust Identification Error, as explained in [**?**].

# 1 Introduction

The RIE toolkit optimizes models of the form:

$$e(x(t+1)) = f(x(t), u(t)) \tag{1}$$
$$y(t) = g(x(t), u(t)) \tag{2}$$

in Discrete Time (DT) and:

$$\frac{d}{dt}e(x(t)) = f(x(t), u(t)) \tag{3}$$
$$y(t) = g(x(t), u(t)) \tag{4}$$

in Continuous Time (CT). Here, $x(t) \in \mathbb{R}^n$ is the system state, $y(t) \in \mathbb{R}^k$ is the system output and $u(t) \in \mathbb{R}^m$ is the system input. We will use $v(t) \in \mathbb{R}^n$ to denote:

$$v(t) = x(t+1) \qquad \text{(DT)}$$
$$v(t) = \dot{x}(t) \qquad \text{(CT)}$$

## 1.1 Quick Start

One function provides an immediate interface to producing RIE optimized fits:

```
[model,P,r,info] = nlid_vxuy2efg(data,domain,stable,...
                                 fmonom,emonom,gmonom,local);
```

The following is a detailed explanation of the inputs and outputs of this function.

### 1.1.1 Inputs:

`data` The toolkit generally works with sampled data matricies. If $s(t) \in \mathbb{R}^p$, then we have a data-matrix:

$$S = \begin{bmatrix} s(t_1) & \dots & s(t_N) \end{bmatrix} \in \mathbb{R}^{p \times N}$$

Our fitting procedure generally deals with four data matricies, $(V, X, U, Y)$ derived from $v(t), x(t), u(t), y(t)$. These matricies are bundled into a `data` struct with `data.V,data.X,data.U,data.Y` holding each matrix.

`domain` A character string, either `'CT'` or `'DT'`.

`stable` A flag, `stable = 1`, indicating if incremental stability should be imposed on the model, that is given two initial conditions $x_1(0) = x_1$ and $x_2(0) = x_2$, and a single output $u_1(t) = u_2(t) = u(t)$, the states converge $\lim_{t\to\infty} |x_1(t) - x_2(t)| = 0$. This is currently supported for DT fits only. The default is `stable = 0`.

`fmonom,emonom,gmonom` Functions returning monomials for each function $(e, f, g)$, e.g.:

```
gmonom = @(x,u) mpmonomials({x,u},{0:3,0:3});
fmonom = @(x,u) mpmonomials({x,u},{0:4,0:1});
emonom = @(x) monomials(x,1:5);
```

`local` A flag, `local = 1` indicating the local RIE should be used. The default is `local = 1`.

### 1.1.2 Outputs:

`model` A structure containing the resulting model. The fields of the structure are:

**domain** The domain of the model (`'CT'` or `'DT'`).

**v,x,u,y** Simple `msspoly` objects representing the update, state, input and output.

**e,f,g** The model, after an affine change of coordinates, represented by `msspoly` objects.

**affine** The affine change of coordinates.

`P` A positive definite matrix involved in the storage function (see [**?**]).

`r` Upper bounds on the RIE for each data-point.

`info` Diagnostic information form the solver.

## 1.2 Simulate the Model

Having identified the model, we now simulate the system.

```
[ts,yfit,xfit] = sim_efg_model(model,ut,tspan,x0);
```

The argument `tspan` should be a monotonically increasing row-vector. For the DT domain, the step sizes should be identical. If the model is in the CT domain, `ut` should be a function of time, otherwise, `ut` should be an $m \times N$ data matrix where $N$ is the length of the row vector `tspan`.