

# Model Reduction Tool for Discrete Function Approximation

Martha A. Gallivan

September 24, 2000

## 1 Purpose

Reduce the dimension of a system of ordinary differential equations by projecting the equations onto a set of orthogonal modes, or basis vectors. The reduced system of equations has a dimension equal to the number of modes considered. In practice, this tool may also be applied to systems of partial differential equations once they have been discretized for simulation.

- Inputs:
  - vector field to be reduced
  - grid on which to compute the values and derivatives of the vector field
  - orthogonal modes associated with the vector field
  - grid for new vector field describing the modes
  - number of derivatives to compute
- Output:
  - new dfa object describing the evolution of the modes

## 2 Equations

Consider a system of ordinary differential equations

$$\dot{x} = f(x) \tag{1}$$

where  $x \in \mathbb{R}^n$  and  $f : \mathbb{R}^n \mapsto \mathbb{R}^n$ . To reduce the dimension of the system, we make the assumption that  $x$  can be represented by  $m$  orthonormal spatial modes  $\phi_j \in \mathbb{R}^n$ , such that

$$x = \sum_{j=1}^m a_j(t) \phi_j \tag{2}$$

where  $a \in \mathbb{R}^m$ . The  $a_j$ 's may be thought of as the time-varying coefficients of the modes. We assume that the spatial modes have already been obtained from numerical simulations of (1). The task is now to compute the time evolution of the  $a_j$ 's. Since there are  $m$  time coefficients, the reduced system will have a dimension of  $m$ .

By substituting (2) into (1), we obtain a system of differential equations for the  $a_j$ 's.

$$\sum_{j=1}^m \dot{a}_j(t) \phi_j = f \left( \sum_{j=1}^m a_j(t) \phi_j \right) \quad (3)$$

Taking the inner product of (3) with each spatial mode and using the orthonormality of the modes, we isolate the  $\dot{a}_j$ 's on the left side.

$$\left\langle \sum_{j=1}^m \dot{a}_j(t) \phi_j, \phi_k \right\rangle = \left\langle f \left( \sum_{j=1}^m a_j(t) \phi_j \right), \phi_k \right\rangle \quad (4)$$

$$\dot{a}_k(t) = \left\langle f \left( \sum_{j=1}^m a_j(t) \phi_j \right), \phi_k \right\rangle \quad (5)$$

Defining the right side of (5) as  $\hat{f}(a)_k$ ,

$$\hat{f}(a)_k = \left\langle f \left( \sum_{j=1}^m a_j(t) \phi_j \right), \phi_k \right\rangle \quad (6)$$

we have a new vector field for the time coefficients of the spatial modes

$$\dot{a} = \hat{f}(a) \quad (7)$$

where  $a \in \mathbb{R}^m$  and  $\hat{f} : \mathbb{R}^m \mapsto \mathbb{R}^m$ .

In addition to the computing the value of the new vector field at each grid point, we also wish to know the value of its derivatives. Because we do not have an analytical representation for  $\hat{f}(a)$ , we must compute its derivatives from the derivatives of the original vector field  $f(x)$ . Using (6), the chain rule

$$\frac{\partial \hat{f}}{\partial a_i} = \sum_{j=1}^n \frac{\partial \hat{f}}{\partial x_j} \frac{\partial x_j}{\partial a_i} \quad (8)$$

and the fact that derivatives of  $x$  with respect to  $a$  are zero for orders higher than 1, we obtain

$$\frac{\partial^p \hat{f}(a)}{\partial a_{i_1} \cdots \partial a_{i_p}} = \sum_{j_1=1}^n \cdots \sum_{j_p=1}^n \frac{\partial^p f}{\partial x_{j_1} \cdots \partial x_{j_p}} \left( \prod_{q=1}^p \phi_{i_q, j_q} \right) \phi \quad (9)$$

where  $p$  is the desired order of the derivative and  $\phi$  is the matrix containing all of the spatial modes.

### 3 Additional Comments

The method in which the derivatives are stored makes the DFA objects difficult to manipulate. This makes it hard to compute derivatives symbolically outside Matlab and read them in. It is also cumbersome to manipulate derivatives inside Matlab. See the portion of my code in which I compute the derivatives of the reduced vector field to see what I mean. Matlab commands such as `eval` and `reshape` help some.

The DFA function `add_box_coeff` seems buggy. I sometimes get memory errors when I run the code. I've traced the error back to the `spline_box_cart` function, which is written in C and is in the `dfa/Ccode` directory.

On non-Solaris platforms, be sure to recompile all the C files in the `dfa/Ccode` directory. Otherwise, Matlab will not know these functions.

## 4 Code

### 4.1 `mr_script.m`

```
addpath /usr/local/dfa
addpath /usr/local/dfa/interpolation
addpath /usr/local/dfa/Ccode

% Model reduction of ODEs
%
% Martha Gallivan 9-19-00

% Inputs: vector field, grid for vector field, modes, grid for modes,
% number of derivatives
% Preprocess to get... dfa object of vector field
% Interpolate to get a dfa object on the new points
% Compute inner product
% Construct the dfa output object
% Output: dfa object of vector field for the modes

% Set the number of derivatives desired
nder = 4;

% Set the number of states of the original vector field
ns = 3;

% Set the domain for the original function
dvh = uniform_box([-1 1; -1 1; -1 1], 3*ones(1,ns));

% Construct the dfa object for the vector field
fd = sample('mr_vf', dvh, nder, 0.001*ones(1,ns));

% Read in the matrix of modes, size ns*nm
load vfmodes.dat
sm = size(vfmodes);
nm = sm(1);

% Define the domain for the modes
dm = uniform_box([-1 1; -1 1], 3*ones(1,nm));
```

```
% Call the main function to compute the vector field for the modes
gd = dfa_mr(fd, vfmodes, dm);
```

## 4.2 dfa\_mr.m

```
function gd = dfa_mr(fd,vfmodes,dm)
%
% Function to perform model reduction on a system of ODES
% within the 'dfa' framework
%
% gd = dfa_mr(fd,vfmodes,dm)
%
% Inputs:
% fd is the dfa object to be reduced
% vfmodes is a matrix containing the modes of the vector
% field within fd onto which the system is to be projected
% dm is the dfa domain for the reduced system
% Outputs:
% gd is a dfa object for the reduced system
%
% Note: function is not generalized for tensors (rank > 1)
%
% Martha Gallivan 9-24-00
%
if nderivs(fd) > 4
    error('Maximum order of derivatives is 4.')
```

end

```
% Determine the number of states
ns = dim_value(fd);
if ns(2) > 1
    error('Function does not handle tensors of rank > 1.');
```

end

```
ns = ns(1);
% Check the size of the mode matrix
nm = size(vfmodes);
if nm(2) ~= ns
    error('Number of states in vector field is not consistent with the number of states in the
else
nm = nm(1);
end
if dim(dm) ~= nm
    error('Number of modes is not consistent with dimension of mode domain.');
```

end

```
% Check for orthogonality of modes
for i = 1:nm
    for j = i+1:nm
```

```

    checker = 0;
    for k = 1:ns
        checker = checker + vfmodes(i,k)*vfmodes(j,k);
    end
    if checker ~= 0
        error('Modes are not orthogonal.')
    end
end
end

% Normalize the modes
for i = 1:nm
    mnorm = 0;
    for j = 1:ns
        mnorm = mnorm + vfmodes(i,j)^2 ;
    end
    mnorm = sqrt(mnorm);
    for j = 1:ns
        vfmodes(i,j) = vfmodes(i,j)/mnorm;
    end
end

% Compute the value of the vector field at each point which corresponds to a
% point in the mode domain dm
projpts = zeros(ns, npoints(dm));
dmpoints = points(dm);
for i = 1:npoints(dm)
    projpts(:,i) = vfmodes'*dmpoints(:,i);
end
dproj = dfa_domain(projpts);
fd = add_box_coeff(fd);
fdproj = dfa_box_interp(fd, dproj, 'box_poly_interp');
% Check for extrapolated points
eflag = 0;
lims = zeros(ns,2);
lims(:,1) = min(points(fd)')';
lims(:,2) = max(points(fd)')';
for i = 1:ns
    if (max(projpts(i,:)) > lims(i,2) | min(projpts(i,:)) < lims(i,1))
        eflag = 1;
    end
end
if eflag
    warning('Domain for modes requires extrapolation of original vector field.');
```

```

end

% Compute the vector field for the modes by taking the inner product
```

```

% of the value with each mode
newvalues = zeros(nm,npoints(dm));
fvalues = values(fdproj);
for i = 1:nm
    for j = 1:npoints(dm)
        newvalues(i,j) = fvalues(:,j) * vfmodes(i,:);
    end
end

% Compute the derivatives of the new vector field
% Ugly code!!!
newderivs = cell(1,nderivs(fd));
for k = 1:nderivs(fd)
    newderivs{k} = zeros([nm 1 nm*ones(1,k) npoints(dm)]);
end
dvalues = derivs(fdproj);
for q = 1:npoints(dm)
    for k = 1:nderivs(fdproj)
        str = 'for j1 = 1:ns';
        for s = 2:k
            str = [str ', for j' num2str(s) ' = 1:ns'];
        end
        str = [str ', for i1 = 1:nm'];
        for s = 2:k
            str = [str ', for i' num2str(s) ' = 1:nm'];
        end
        str = [str ', pimodes = vfmodes(i' num2str(1) ',j' num2str(1) ')'];
        for s = 2:k
            str = [str '*vfmodes(i' num2str(s) ',j' num2str(s) ')'];
        end
        str = [str '; dummy = ctranspose(pimodes*ctranspose(dvalues{k}(:,j1)'];
        for s = 2:k
            str = [str ',j' num2str(s)];
        end
        str = [str ',q)*ctranspose(vfmodes));'];
        str = [str 'newderivs{' num2str(k) '}(:,i1)'];
        for s = 2:k
            str = [str ',i' num2str(s)];
        end
        str = [str ',q) = newderivs{' num2str(k) '}(:,i1)'];
        for s = 2:k
            str = [str ',i' num2str(s)];
        end
        str = [str ',q) + dummy;'];
        for s = 1:k
            str = [str ',end,end'];
        end
    end
end

```

```
        eval(str)
    end
end

% Create a new dfa object for the new vector field
gd = dfa_object(dm, newvalues, newderivs);
```

### 4.3 mr\_vf.m

```
%
% Sample function for use with mr_script.m and dfa_mr.m
%
% Martha Gallivan 9-24-00
%
function flowin = mr_vf(x)

flowin = [x(1) + x(2); x(1)^2; x(3)];
```

### 4.4 vfmodes.dat

```
%
% Sample file for mr_script.m
%
% Contains the spatial modes onto which a vector field is projected.
% Each row represents a mode.
%
% Martha Gallivan 9-24-00
%
1 0 -4
0 1 0
```