Generation of right-hand sides and their derivatives with Matlab's symbolic toolbox — `symcoco`

Jan Sieber, University of Exeter

Contents

The routines in `symcoco` provide a wrapper around Matlab's symbolic toolbox and its code generation functions to generate user-defined right-hand sides and their derivatives in a vectorized form.

Matlab's symbolic toolbox only needs to be present during the generation of the right-hand sides. No COCO routines are called during the generation. The generated functions use only standard Matlab commands such that access to the symbolic toolbox is not required during COCO computations.

The symbolic operations, including the call to `sco_sym2funcs` is compatible with `octave`'s `symbolic` package.

1 Changes

(2023)

- (Bugfix) The symbolic toolbox's `matlabFunction` may create local subfunctions on its own , always using the same name. This prevents utting all generated functions into one file. A work-aound is the new option `'multifile'` and `'folder'`, where each generated function is put into its own file, if requested into the requested folder (`'./private` is a convenient choice).

- (Bugfix) The symbolic toolbox's `matlabFunction` may treat symbolic expressions that overlap builtins incorrectly (e.g., `beta`). A new default is to rename all variables to `in1`,`in2`,.... Set prefix with option `'inname'` and disable this renaming by setting option `'rename'` to `false` (default `true`).

- (Bugfix) The function `sco_sym2funcs` tests if `matlabFunction` generates functions with the `end` keyword (this is not the case for versions before 2021). This avoids syntax errors in the function file generated by `sco_sym2funcs`.

- Function `sco_gen` can now output full derivatives up to arbitrary order. The maximal order is set by the optional `maxorder` argument in `sco_sym2funcs`. These can be mixed with directional derivatives.

2 Typical usage

2.1 Detailed demo based on EP toolbox demo `bistable`

See folder `examples_doc/bistable` for
- script `gen_sym_bistable.m` and its html output `bistable/html/gen_sym_bistable.html`: a script demonstrating how one may generate right-hand sides and their partial derivatives using the symbolic toolbox and wrapper `symcoco`,
- script `demo.m` and its html output `bistable/html/demo.html`: a script demonstrating how one may call the functions generated with `symcoco`, and use them for COCO computations.

The example `bistable` is copied from the `po`-toolbox demo of same name. The first part of the demo demonstrates the user interface for `symcoco` in detail, before following the original `ep`-toolbox demo, using the symbolic derivatives.

2.2 Modification of demos `sphere_optim` and `int_optim`

Tasks involving optimization need second partial derivatives of all constraints. The demos `sphere_optim` from CORE-Tutorial and `int_optim` from the PO-Tutorial have been modified to demonstrate how derivatives for these problems can be constructed using `symcoco`.

sphere_optim The outputs sphere_optim/html/demo.html and sphere_optim/html/gen_sym_sphere.html demonstrate construction using sco_sym2funcs and usage using sco_gen of constraints and objective functional. This demo shows how one can put the simple functions generated by sco_gen into the COCO core format

```
function [data,y]=func(prob,data,u)
```

by using

```
fcn = @(f) @(p,d,u) deal(d, f(u));
obj=sco_gen(@sym_sphere_obj);
funcs2 = { fcn(obj('')),fcn(obj('u')),fcn(obj({'u','u'}))};
```

int_optim The outputs int_optim/html/demo.html and int_optim/html/gen_sym_int_optim.html demonstrate construction using sco_sym2funcs and usage using sco_gen of constraints and objective functional of a PO-toolbox demo for successive continuation with integral objective functional and ODE constraints.

3 Call arguments

Construction happens in two steps. Only the first step requires the symbolic toolbox (or, alternatively, the symbolic package in octave, based on the python package sympy. Each step is a function call:

1. sco_sym2funcs during generation of the right-hand side and its derivatives using the symbolic toolbox;
2. sco_gen for creating wrappers around the function created by sco_sym2funcs that can be used for COCO computations.

Function input/output formats:

```
function [fout,funcstr,derivatives]=sco_sym2funcs(f,args,names,varargin)
function fout=sco_gen(fun,name)
```

where varargin are optional pairs of the form 'name',value with defaults. Both calls in combination convert symbolic expression f into a matlab function of the form

```
function y=sys(action,argseq)
```

where y is a $n_y \times 1$ vector, and *argseq* is a sequence of n_a arguments $argseq_i$ of shape $n_i \times 1$, such that *sys* depends on overall $n_u = \sum_{i=1}^{n_a} n_i$ scalar variables. Only sco_sym2funcs depends on the symbolic toolbox. It is a wrapper around matlabFunction producing a function file (by default sys.m), which contains all intermediate information for the wrapper sco_gen to return *sys* and its derivatives.

3.1 Example of `sco_symfuncs` call

Demo `bistable` creates a function

$$\mathbb{R} \times \mathbb{R}^2 \times \mathbb{R}^3 \ni (t, x, p) \mapsto f(t, x, p) \in \mathbb{R}^2$$

and its derivatives up to order 3. This is the underlying right-hand side f of a differential equation,

$$\begin{aligned} \dot{x} &= v, \\ \dot{v} &= -\gamma v - x - x^3 + a\cos(2\pi t/T), \end{aligned}$$

where we collect $(x; v) \in \mathbb{R}^2$ into the state vector (also called x) and $(T; a; \gamma)$ into the parameter vector $p \in \mathbb{R}^2$:

$$f\left(t, \begin{bmatrix} x \\ v \end{bmatrix}, \begin{bmatrix} T \\ a \\ \gamma \end{bmatrix}\right) = \begin{bmatrix} v \\ -\gamma v - x - x^3 + a\cos(2\pi t/T) \end{bmatrix}.$$

```
syms t x v gam a T
f=[v; -gam*v-x-x^3+a*cos(2*pi*t/T)];
F=sco_sym2funcs(f,...                 % symbolic expression for f
    {t,[x;v],[T;a;gam]},...          % which symbols are in which inputs of f
    {'t','x','p'},...                % names for inputs of f
    'vector',[0,1,1],...             % are inputs scalar or vectors
    'filename','sym_bistable',... % filename for result
    'maxorder',3);                   % derivatives are computed up to this order
```

The resulting f is stored in `sym_bistable.m` and function handles accessing it are generated with `sco_gen`. See the calling examples in section 3.6 for the effect of the optional argument `'vector'`.

3.2 Examples of `sco_gen` usage

See demo `test`. Use symbolic toolbox to create a shortcut for generating derivatives:

```
syms t x v gam a T
F=sco_sym2funcs([v; -gam*v-x-x^3+a*cos(2*pi*t/T)], {t,[x;v],[T;a;gam]},...
   {'t','x','p'}, 'vector',[0,1,1], 'filename','bistable', 'maxorder',3);
```

Output `F` can alternatively be generated after call to `sco_sym2funcs` by

```
F=sco_gen(@bistable);    % same as above F
```

Suppose the arrays $t \in \mathbb{R}^{1\times N}$, $x \in \mathbb{R}^{n_x \times N}$ and $p \in \mathbb{R}^{n_p \times 1}$ have been created (with $n_x = 2$, $n_p = 3$). The right-hand side and first derivatives are generated and called as follows:

```
  f=F('');            fx=F('x');
 fxp=F({'x','p'});   fpx=F({'p','x'});
fxvp=F({'x*v','p'}); ftx=F({'t','x'});
 df3=F(3);           df3c=F({3})
dfxpdir=@(t,x,p,dt,dx,dp)df3c(t,x,p,{0,0,dt},{'I',0,dx},{0,'I',dp})
```

`f(t,x,p)`:	$f(t,x,p)\in\mathbb{R}^{n_y\times N}$	
`fxp(t,x,p)`:	$\partial^2_{xp}f(t,x,p)\in\mathbb{R}^{n_y\times n_x\times n_p\times N}$	
`fpx(t,x,p)`:	$\partial^2_{px}f(t,x,p)\in\mathbb{R}^{n_y\times n_p\times n_x\times N}$	(note effect of ordering in tensor)
`fxvp(t,x,p,v)`:	$\partial^2_{xp}f(t,x,p)v\in\mathbb{R}^{n_y\times n_p\times N}$	
`ftx(t,x,p)`:	$\partial^2_{tx}f(t,x,p)\in\mathbb{R}^{n_y\times n_x\times N}$	(as `isv`$_1$ is `false`, **not** in $\mathbb{R}^{n_y\times 1\times n_x\times N}$).

A mix of directional derivatives and full derivatives:

`dfxpdir(t,x,p,dt,dx,dp)`:

$$\partial^2_{xp}\left[\partial_t f(t,x,p)\delta_t+\partial_x f(t,x,p)\delta_x+\partial_p f(t,x,p)\delta_p\right]\in\mathbb{R}^{n_y\times n_x\times n_p\times N}.$$

Total derivatives ($n_u=1+n_x+n_p$, $u=(t,x,p)$, $\delta_i=(\delta_{t,i},\delta_{x,i},\delta_{p,i})$):

`df3(t,x,p)`:	$\partial^3 f(u)$	$\in\mathbb{R}^{n_y\times(n_u)^3\times N}$
`df3(t,x,p,{dt1},{dx1},{dp1})`:	$\partial^3 f(u)\delta_1$	$\in\mathbb{R}^{n_y\times(n_u)^2\times N}$
`df3(t,x,p,{dt1,dt2},{dx1,dx2},{dp1,dp2})`:	$\partial^3 f(u)\delta_1\delta_2$	$\in\mathbb{R}^{n_y\times n_u\times N}$
`df3(t,x,p,{dt1,dt2,dt3},{dx1,dx2,dx3},{dp1,dp2,dp3})`:	$\partial^3 f(u)\delta_1\delta_2\delta_3$	$\in\mathbb{R}^{n_y\times N}$

3.3 Inputs of `sco_sym2funcs`

- `f`: right-hand side, $n_y\times 1$ array of matlab symbolic expressions. This variable contains the mathematical expression to be converted into the matlab function *sys* and differentiated with respect to its arguments. The expression `f` depends on n_u scalar symbolic variables (a scalar symbolic variable `x` is a Matlab variable for which `diff(f,x)` is a valid expression).
- `args`: partition of variables in symbolic expression `f` into arguments *argseq* of function *sys*, $1\times n_a$ cell array. Each element `args{i}` is a $n_i\times 1$ array of scalar symbolic variables, corresponding to *argseq*$_i$, such that `cat(1,args{:})` is an array of length n_u containing all scalar variables that `f` depends on.
- `names`: names of arguments of output function *sys*, $1\times n_a$ cell array of character strings. These names can be used in second input of `sco_gen` to obtain partial derivatives with respect to arguments of *sys*.
- (optional) `'filename'` (character string, default `'sys'`): file name (without ending `'.m'`), in which the resulting function is stored as a side effect of `sco_sym2funcs`.

- (optional) `'vector'` (logical $1 \times n_a$ vector `isv`, default `true(1,length(args))`): flag whether `args{i}` is treated as a scalar or vector. Note that if `isv(i)` is false then n_i must be equal to 1, but `isv(i)` may be true, even if $n_i = 1$. This affects the vectorized output of partial derivatives with respect to $argseq_i$.
- (optional) `'maxorder'` (positive integer, default 2): maximal order up to which derivatives will be computed. Currently, the wrapper `sco_gen` provides only directional derivatives for derivatives of order greater than 2.
- (optional) `'write'` (logical, default `true`): instruction whether to write the resulting function to file.
- (optional) `'multifile'` (logical, default `false`). When the code generation is more complex, `matlabFunction` may generate local subfunctions with hard-to-control names. This prevents appending the generated functions from being collected in a single file. With this option , the generated files will be of the form *filename*`_rhs_`k`.m`, where *filename* is the optional input for `'filename'` and k is the order of the derivative.
- (optional) `'folder'` (character string, default `pwd()`) the generated files may be placed in a different folder, provided here. A convenient choice may be the local subfolder `fullfile(pwd(),'private')`, because then the generated files will be found by all script or function files in the current folder (however, beware that they are not found on the command line).
- (optional) `'output'` (one of the strings from `'fout'`, `'funcstr'` and `'derivatives'`, or a cell with several of these strings, default `{'fout','funcstr','derivatives'}`) controls the order and appearance of outputs.

3.4 Outputs of `sco_sym2funcs`

The main result is usually the side effect producing a file storing the generated functions. Control order and appearance of outputs using optional input `'output'`

- `fout` if option `'write'` is set to true, `sco_sym2funcs` calls `fout=sco_gen(str2func(filename))` at the end of its code generation to create the function generator `fout` (see outputs of `sco_gen` in section 3.6 for how to use `fout`). Note that this output cannot be generated if function files are stored in a `private` folder.
- `funcstr` character array (including newlines for line breaks that contains the text that was (or would have been) written to the output file.
- `derivatives` array of structures of length `maxorder-1`. The structure `derivatives(i+1)` contains the fields `df` of shape $n_y \times 1$, and `x` and `dx` (both of shape $n_u \times 1$). The symbolic expression `derivatives(i).df` contains the directional derivative of order $i-1$ in `x`, in direction `dx`. The field `x` will equal `cat(1,args:)`, the field `dx` contains the names of the deviations (by default these are the names in `x`, extended by `'_dev'`).

3.5 Inputs of `sco_gen`

- `fun`: character string, name of file or function handle produced by `sco_symfuncs`.

- `name` has several use cases. It controls the format and nature of the output `fout`. Name may be
  (i) absent, or
  (ii) the empty string (`''`),
  (iii) a character string from the list in input *names* of `sco_sym2funcs`,
  (iv) $1 \times m$ cell of character strings from the list in input *names* of `sco_sym2funcs` ($1 \leq m \leq$`'maxorder'` argument from `sco_sym2funcs`),
  (v) each of the character strings in points iii or iv may be followed by a `'*'` and an arbitrary (ignored) letter sequence, or
  (vi) integer `k` less or equal than optional input `'maxorder'` of `sco_sym2funcs`.
- `debug` (default `false`): logical flag. If set to `true`, all assertions concerning argument format inside the generated functions are tested (slowing the functions down).

See Section 3.6 for details and examples.

3.6 Output of `sco_gen`

Output `fout` is a function handle that can be called in a vectorized form, returning the function *sys* or its partial derivatives with respect to the arguments in *argseq*, or its directional derivatives. The format of `fout` depends on the second input, `name` to `sco_gen`.

(i) (`name` absent)
`F=sco_gen(fun);` returns `F=@(name)sco_gen(fun,name,false);`
`F=sco_gen(fun,'_debug');` returns `F=@(name)sco_gen(fun,name,true);`
to provide shortcuts. The final logical flag switches on assertion checking.

(ii) (`name` is `''`) `fout` is handle to *func*, expecting n_a arguments, where the ith argument has shape $n_i \times N$, for some $N \geq 1$. After `y=fout(...)`, y has shape $n_y \times N$.

(iii) (`name` is character string *argname* or 1×1 cell with a character string *argname* from the list in input *names* of `sco_sym2funcs`) `fout` is function handle, expecting n_a arguments, where the ith argument has shape $n_i \times N$, for some $N \geq 1$. If *argname* equals names_k, then, after `J=fout(...)`, J is the partial derivative of *sys* with respect to argseq_k which has shape $n_y \times n_k \times N$.

(iv) (`name` is $1 \times m$ cell{ $\mathit{argname}_1, \ldots \mathit{argname}_m$} with character strings from the list in input *names* of `sco_sym2funcs`) `fout` is function handle, expecting n_a arguments, where the ith argument has shape $n_i \times N$, for some $N \geq 1$. If $\mathit{argname}_j$ equals names_{k_j} and $\mathtt{isv}_{k_j}$ is `true`, then, after `J=fout(...)`, J is the m-order partial derivative of *sys* with respect to $\mathit{argseq}_{k_1}, \ldots, \mathit{argseq}_{k_m}$, which has shape $n_y \times n_{k_1} \times \ldots \times n_{k_m} \times N$.

(v) Arguments in points (iii) and (iv) may contain the symbol `'*'` to create directional derivatives in this argument. Then `fout` requires one additional argument and the output reduces by the dimension of this input.

(vi) (name is integer `k` less than or equal to the optional input `'maxorder'` of `sco_sym2funcs`) `fout` is function handle to the total derivative of order k, applied to up to k deviations, expecting n_a or $2n_a$ inputs. Input i for $i \leq n_a$ is double array of shape $n_i \times N$, corresponding to the base point where the total or directional derivative of *sys* is taken.

Input n_a+i for $i \in \{1,\dots,n_a\}$ is a cell array of length $\ell \le k$, where each entry has shape $n_i \times N$. These are the deviations for the kth derivative applied to the ith argument of *sys*.

(vii) (name is cell containing single integer `k` less than or equal to the optional input `'maxorder'` of `sco_sym2funcs`) Same as integer argument, but all deviation arguments must have k elements. For this call optionally, the ℓth entry of one cell array argument may be the capital letter `'I'`. Then all other ℓth entries of all cells have to be `0`. For the entry with `'I'` the derivative will be full.

4 Internal procedure for differentiation and code generation

4.1 Procedure for symbolic differentiation and code generation

In short, the only functionality of the Matlab symbolic toolbox that is needed are `subs`, `diff` and the code generation tool `matlabFunction`. Step-by-step procedure is described below.

1. Inside `sco_sym2funcs` all variables, which the symbolic expression f (`f`) depends on are collected as a single argument u =`vertcat(args{:});`.
2. New symbols u_{dev} (`udev`) are introduced, with names (by default) `[x,'_dev']`, where `x` are the names of the symbols in u. The name of the extension can be overwritten by the optional input `'dev_append'` to `sco_sym2funcs`.
3. A new symbol h (`h`) with default name `'h_devsmall'` is introduced. The name of h can be overwritten by the optional input `'deviation_name'` to `sco_sym2funcs`.
4. Derivatives up to order $k_{\max}$ (given by optional input `'maxorder'`, default $k_{\max}=2$) of the form
$$D_k f(u)[u_{\mathrm{dev}}]^k := \left.\frac{\partial^k}{\partial h^k} f(u+hu_{\mathrm{dev}})\right|_{h=0} \tag{1}$$
are computed via symbolic toolbox commands
```
fdev=subs(f,u,u+h*udev);
hrep=repmat({h},1,k);
df{k+1}=subs(diff(fdev,hrep{:}),h,0);
```
where the new symbolic expressions `df{k}` depend on the symbolic variables `[u(:);udev(:)]`, and `df{1}` is the original expression `f`.
5. Each expression `df{k}` is output (for Matlab) into a temporary file
```
folder=tempname;
fname=sprintf('%s_%d',[filename,'_rhs'],k-1);
filename=fullfile(folder,[fname,'.m']);
```
where `filename` is the optional input `'filename'` of `sco_sym2funcs` (default `'sys'`), using symbolic toolbox function `matlabFunction`.
For octave, the expressions are passed on to `sympy.utilities.codegen.codegen` in the python module `sympy`.

6. The temporary files are read back into a character string, to which a header, providing control for calling the resulting intermediate function file, is added. The resulting character string is then written to the optional input `'filename'` of `sco_sym2funcs` (default `'sys'`), with extension `'.m'`. By default, the temporary files are deleted (overwrite with optional argument `'keeptemp'`).

Directional derivatives in arbitrary directions The expression (1) constructs kth order derivatives only in a single direction u_{dev}. Directional kth derivatives with an arbitrary set of k directions, $\partial^k f(u)[v_1,\ldots,v_k]$ are then constructed using the telescope formula

$$\partial^k f(u)[v_1,\ldots,v_k] = \frac{1}{2^{k-1}(k!)} \sum_{p\in\{-1,1\}^k} \left[\prod_{i=1}^{k} p_i\right] \partial^k f(u)\left[\sum_{i=1}^{k} p_i v_i\right]^k.$$

The index set $\{-1,1\}^k$ refers to all sequences p of -1's and 1's of length k. The special case for $k=1$ is trivial ($\partial^1 f(u)v_1$). For the case for $k=2$ the formula equals

$$\frac{1}{4}\left(\partial^2 f(u)[v_1+v_2]^2 - \partial^2 f(u)[v_1-v_2]^2\right).$$

4.2 Control information and calling format of generated intermediate function

The call to `sco_sym2funcs` creates a matlab function with default name `sys` of the format

```
function varargout=sys(action,varargin)
```

The input `action` is a character string that controls the type of output:

- `'nargs'`: number of arguments, n_a,
- `'nout'`: row dimension of output, n_y,
- `'argrange'`: structure with field names equal to names_i, and values pointing into the range or rows of input `u=cat(1,args{:})`. Thus, `argrange.(`names_i`)` is the range of indices from $1+\sum_{j=1}^{i-1} n_j$ to $\sum_{j=1}^{i} n_j$.
- `'argsize'`: structure with field names equal to names_i, and values n_i.
- `'vector'`: $1\times n_a$ logical array, equal to optional input `'vector'`.
- `'maxorder'`: integer, maximal order of derivatives computed, equal to input `'maxorder'`.
- `'extension'`: extension of name for the functions for the directional derivatives (the name is equal to `[`*filename*`,'_',`*extension*`,'_',num2str(`k`)]` for the $k$the derivative. This string is needed for calling the function (see next item).
- If `action` equals the string returned by `sys('extension')`, then the kth derivative in u in direction u_{dev} is returned. The integer k is `varargin{1}` ($k=0$ is possible and returns the undifferentiated function). The row i of u is `varargin{1+i}`, row i of u_{dev} is `varargin{1+`n_u`+i}`.