# 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.

## 1 Typical usage

### 1.1 Detailed demo based on EP toolbox demo `bistable`

See folder `symcoco_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.

### 1.2 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.

## 2 Call arguments

The user interacts typically with two functions:

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

Function input/output formats:

```
function [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(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 produces a function file (by default `sys.m`), which contains all intermediate information for the wrapper `sco_gen` to return $sys$ and its derivatives.

### 2.1 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.

### 2.2 Outputs of `sco_sym2funcs`

(often not needed)

- `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'`).

### 2.3 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
  - absent
  - `''`,
  - a character string from the list in input *`names`* of `sco_sym2funcs`,
  - $1 \times 1$ or $1 \times 2$ cell of character strings from the list in input *`names`* of `sco_sym2funcs`, or
  - integer `k` less or equal than optional input `'maxorder'` of `sco_sym2funcs`.

  See Section 2.4 for details.

### 2.4 Output of `sco_gen`

Output `fout` is a function handle that an be called in a vectorized form, returnign 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`.

- (`name` absent) `F=sco_gen(fun);` is shortcut for `F=@(varargin)sco_gen(fun,varargin:);`
- (`name` is `''`) `fout` is handle to *`func`*, expecting $n_a$ arguments, where the $i$th argument has shape $n_i \times N$, for some $N \geq 1$. After `y=fout(...)`, $y$ has shape $n_y \times N$.
- (`name` is character string *`argname`* or $1 \times 1$ 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 $i$th argument has shape $n_i \times N$, for some $N \geq 1$. If *`argname`* equals $\mathit{names}_k$, then, after `J=fout(...)`, $J$ is the partial derivative of *`sys`* with respect to $\mathit{argseq}_k$ which has shape $n_y \times n_k \times N$.
- (`name` is $1 \times 2$ cell $\{\mathit{argname}_1, \mathit{argname}_2\}$ with character strings from the list in input *`names`* of `sco_sym2funcs`) `fout` is function handle, expecting $n_a$ arguments, where the $i$th argument has shape $n_i \times N$, for some $N \geq 1$. If $\mathit{argname}_j$ equals $\mathit{names}_{k_j}$, then, after `J=fout(...)`, $J$ is the second-order partial derivative of *`sys`* with respect to $\mathit{argseq}_{k_1}$ and $\mathit{argseq}_{k_2}$, which has shape $n_y \times n_{k_1} \times n_{k_2} \times N$.
- (name is integer `k` less or equal than optional input `'maxorder'` of `sco_sym2funcs`) `fout` is function handle to directional derivative of order $k$, expecting $2n_a$ inputs. Input $i$ for $i \leq n_a$ is double array of shape $n_i \times N$, corresponding to base point where directional derivative of *`sys`* is taken. Input $n_a + i$ for $i \in \{1, \ldots, n-a\}$ is a cell array of length $k$, where each entry has shape $n_i \times N$. These are the deviations for the $k$th derivative

applied to the $i$th argument of `sys`.

# 3 Internal procedure for differentiation and code generation

## 3.1 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 $u_{\mathrm{dev}}$ (`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. Then 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
$$D_k f(u)[u_{\mathrm{dev}}]^k := \frac{\partial^k}{\partial h^k} f(u + h u_{\mathrm{dev}})\Big|_{h=0} \tag{1}$$
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 $k$th order derivatives only in a single direction $u_{\text{dev}}$. Directional $k$th 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

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

### 3.2 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 $k$th derivative in $u$ in direction $u_{\text{dev}}$ 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 $u_{\text{dev}}$ is `varargin{1+`$n_u$`+i}`.