Description of the MATLAB functions SENS_SYS and SENS_IND.

V. M. García Mollá*[a] , R. Gómez Padilla[b]
[a]Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, SPAIN
[b]I.E.S. Meliana (Valencia), SPAIN

June 12, 2002

Abstract

The functions SENS_IND and SENS_SYS are extensions of the ODE15s Matlab ODE/DAE solver (version 5.3). These new functions solve ODE/DAE systems while, at the same time, calculate derivatives (sensitivities) of the solution with respect to parameters.

These functions are still under development and must be used with caution.

1 Introduction

The general form of an ODE/DAE system is:

$$\begin{aligned} F(t,y,y') &= 0 \qquad (1)\\ y(t_0) &= y0; \qquad (2)\end{aligned}$$

This system shall be a DAE if $\frac{\partial F}{\partial y'}$ is singular. The Matlab stiff ODE/DAE solver ODE15s [1](written by Mark W. Reichelt, Lawrence F. Shampine, and Jacek Kierzenka[4] [5]) can solve a wide subset of this kind of problems, those that can be stated in the form:

$$\begin{aligned} M(t)y' &= f(t,y) \qquad (3)\\ y(t_0) &= y0;\end{aligned}$$

In this case, it would be a DAE if matrix $M(t)$ is singular.

ODE/DAE systems with parameters appear quite often in engineering :

$$\begin{aligned} M(t)y' &= f(t,y,u) \qquad (4)\\ y(t_0) &= y0;\end{aligned}$$

and it is usually desired to study the dependence of the solution with respect to the parameters:

*Corresponding author: vmgarcia@dsic.upv.es

Example 1 *The Gas-Oil cracking problem*

$$\begin{pmatrix} y_1' \\ y_2' \end{pmatrix} = \begin{pmatrix} -(u_1+u_3)y_1^2 \\ u_1 y_1^2 - u_2 y_2 \end{pmatrix},$$
$$\begin{pmatrix} y_1(0) \\ y_2(0) \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

This is a ODE system with three parameters.

It is also common that some experimental data is available, and what is needed is to find the real (or "best") value of the parameter, so that the solution fits the experimental data (**parameter estimation problem**). It can be desired as well to study the dependence of the solution with respect to initial values, which can be used to solve **boundary value problems.**

There are many real problems where it is necessary to obtain the derivatives of the solution of an ODE/DAE system with respect to some extra parameters. The subroutines SENS_IND and SENS_SYS have been written to perform this task. Both have been written "over" the ODE15s solver (in order to keep its excellent features), and both have the same calling sequence (very similar to that of ODE15s). They differ in the algorithm used to compute the derivatives. SENS_IND uses the "Internal Numerical Differentiation " approach, described first by H.G. Bock in [2] and in deep detail by D. Leineweber [3]; SENS_SYS uses the iterative approximation based on directional derivatives, in a similar form to that used by T. Maly and L. R. Petzold [6].

Both perform the same task, but, as a general rule, SENS_IND is faster and SENS_SYS can obtain more accurate results.

Next, the use of these subroutines shall be described. It would be very useful to the possible user to learn first about ODE15s, since most arguments are identical.

2 Use of SENS_SYS and SENS_IND

Both functions have the same calling sequence, so that we will only comment SENS_SYS. The simplest way of calling SENS_SYS is:

>>*[T,Y,DYDU] = SENS_SYS(ODEFILE,TSPAN,Y0,OPTIONS, U)*

The arguments shall be described next, mentioning at the beginning whether the argument is new or "inherited " from ODE15S:

INPUT ARGUMENTS:

- ODEFILE (ODE15s): This argument must contain the name of an m-file (function) of Matlab which contains the definition of the system to be integrated. For the gas_oil problem, we may have the following odefile, which would be saved as **gas_oil.m**:

```
function Y=gas_oil(t,X,flag,U)
Y(1)=(-U(1)-U(3))*X(1)^2;
Y(2)=U(1)*(X(1)^2)-U(2)*X(2);
Y=Y';
```

For details about writing odefiles, see ODE15s help or [1].

- TSPAN (ODE15s): This must be a vector containing, at least, the time interval for the integration. If solutions (and derivatives) are needed at specific integration points (T0, T1, T2, ..., T$_{end}$), then the vector must be [T0, T1, T2, ..., T$_{end}$].
- Y0 (ODE15s) Initial values.
- OPTIONS(ODE15s) Through this parameter the user can control the default integration parameters in ODE15s and the other MATLAB solvers: absolute and relative tolerances, vectorization of the m-files, whether the system may be a DAE or not, etc. check the ODE15s help for details. Usually all these options have default values.
- U (Optional in ODE15s, necessary in SENS_SYS) This argument must be a column vector containing the parameters.

OUTPUT ARGUMENTS:

- T (ODE15s) Times at which the solution is given.
- Y (ODE15s) the solution, calculated at the times in T.
- DYDU(SENS_SYS). Array, possibly three-dimensional, containing the derivatives of the solution Y with respect to the parameters in U.

If the T vector has NT rows (or time points), the system defined in ODEFILE has NY equations and unknowns, and the U vector has NU parameters, then the DYDU array will have dimensions $NT \times NY \times NU$.

In the gas-oil example, taking a integration interval from 0 to 4, and taking as values of the parameters $u = \begin{pmatrix} 0.9875 & 0.2566 & 0.3323 \end{pmatrix}^T$, the system solution and the derivatives can be obtained calling SENS_SYS (or SENS_IND):

>>*[tn,yn,dydu]=sens_sys('gas_oil',[0 4],[1 ;0],[],[0.9875;0.2566;0.3323]);*

Since no options have been set, the OPTIONS argument is left empty. Once executed, the array *DYDU* will have three dimensions; the first shall be the number of time points, the second the number of equations of the system and the third the number of parameters; in this case, $NT \times 2 \times 3$. For example, we may plot the derivatives of $Y(1)$ and $Y(2)$ with respect to $U(1)$.

>>*plot(tn, dydu(:,:,1))*.

If the output is wanted at certain time points (for example, 0:0.5:4), the call would be:

>>*[tn,yn,dydu]=sens_sys('gas_oil',0:0.5:4,[1 ;0],[],[0.9875;0.2566;0.3323]);*

2.1 Vectorization

As in ODE15s, the integration shall be faster if the m-file (ODEFILE) is coded in a "vectorized" manner, so that f(t,y,flag,[U1,...,Un]) returns [f(t,y,flag,U1), .., f(t,y,flag,Un)]. The input parameter VECPAR must be used to inform SENS_IND that the input odefile is "vectorized".

>>*[T,Y,DYDU] = SENS_SYS(ODEFILE,TSPAN,Y0,OPTIONS,U,VECPAR)*

It must be set to 1 if the odefile is vectorized, and to 0 if it is not. The default is 0 (Not vectorized). Again using the same example, if the odefile is vectorized:

```
function Y=gas_oilv(t,X,flag,U)
Y(1,:)=(-U(1,:)-U(3,:)).*X(1,:).^2;
Y(2,:)=U(1,:).*(X(1,:).^2)-U(2,:).*X(2,:);
```

Then, the solution and the sensitivities can be found (faster) with the call:

```
>>[tn,yn,dydu]=sens_sys('gas_oilv',0:0.5:4,[1 ;0],[],[0.9875;0.2566;0.3323],1)
```

Moreover, since the vectorization procedure is the same as in ODE15s, in the OPTIONS argument the vectorized option may be set to 'on' (See ODESET help); as an example of use of ODESET, the tolerances are increased to $1e-8$ and the 'Vectorized' option is set to 'on':

```
>>opt=odeset('AbsTol',1e-8,'RelTol',1e-8,'Vectorized','on');
>>[tn,yn,dydu]=sens_sys('gas_oilv',0:0.5:4,[1 ;0],opt,[0.9875;0.2566;0.3323],1)
```

However, under some circumstances discussed below, SENS_IND and SENS_SYS will fail with vectorized odefiles.

2.2 Dependence of initial values with respect to parameters

Some times the initial values depend on the parameters as well; a common case would the need to study derivatives of the solution with respect to the initial values.

The default initial values of the derivatives is zero. But when the derivatives of the initial values with respect to the parameters are nonzero, a new argument (INITFILE) must be used:

```
>>[T,Y,DYDU] = SENS_SYS(ODEFILE,TSPAN,Y0,OPTIONS, U, VECPAR,INITFILE)
```

This new argument must be the name of a function that takes as input arguments the initial values and the parameters vector, and must return the derivatives of the initial values with respect to the parameters. Let us suppose that we want to calculate the derivatives of the solution of the gas-oil problem with respect to the initial values of the dependent variables (set to [1;1]). To keep the values of the parameters, we might rewrite the odefile including the numerical values:

```
function Y=gas_oil_ini(t,X,flag,U)
Y(1)=(-0.9875-0.3323)*X(1)^2;
Y(2)=0.9875*(X(1)^2)-0.2566*X(2);
Y=Y';
```

Note that in this case U is a dummy argument.

Then, we may write an INITFILE, quite trivial in this case:

```
function Y=ini_gas(X,U)
```

Y=eye(2);

Of course, the derivatives of the initial values with respect to themselves form the identity matrix. Finally, we can call the SENS_SYS subroutine:

```
>>[tn,yn,dydu]=sens_sys('gas_oil_ini',0:0.5:4,[1 ;0],[],[1;1],0,'ini_gas')
```

At present, the subroutine SENS_SYS fails when (in cases like this) the parameter vector does not have real influence and vectorization is used. The SENS_IND subroutine does not have this problem and allows vectorization in these cases.

2.3 Extra Parameters

There may be extra parameters in the system, and we may not be interested in calculating the derivatives with respect to them. In this case, we may pass them as extra parameters (PE1, PE2, ...):

```
>>[T,Y,DYDU] = SENS_SYS(ODEFILE,TSPAN,Y0,OPTIONS,U, VECPAR, INITFILE,PE1,PE2,...)
```

As an example of this feature, the former example can be reformulated so that we do not have to plug the numerical values of the "old" parameters in the odefile, but we can pass them as these "extra" parameters. Rewrite the odefile as:

```
function Y=gas_oil2(t,X,flag,V,U)
Y(1)=(-U(1)-U(3))*X(1)^2;
Y(2)=U(1)*(X(1)^2)-U(2)*X(2);
Y=Y';
```

Again, U is a dummy argument (in this case, where we want to study the dependence with respect to the initial values). Now, the call would be:

```
>>[tn,yn,dydp]=sens_sys('gas_oil2',[0 4],[1;0],[],[1;1],0,'ini_gas', [0.9875; 0.2566; 0.3323])
```

This feature is not compatible with vectorization; if extra parameters are passed, VECPAR must be set to zero and the odefile must be coded accordingly.

2.4 Solving a simple boundary value problem by "shooting"

As an example of possible use of SENS_SYS and SENS_IND, we will show how to solve an easy two-point boundary value problem, proposed in [7]:

$$u'' + e^{u+1} = 0$$
$$u(0) = u(1) = 0$$

This problem must be reformulated to the standard first order form:

$$u_2' = u_1$$
$$u_1' = -e^{u_2+1}$$
$$u_2(0) = u_2(1) = 0$$

Here, $u_1(0)$ is unknown. To solve this problem through single shooting, we need first to write an odefile for the first order form of the problem:

```
function y=bvp(t,x,flag,U)
y(1)=-exp(x(2)+1);
y(2)=x(1);
y=y';
```

Then, an INITFILE for the derivatives of the dependent variables with respect to the initial values at the initial time:

```
function Y=ini_bvp(X,U)
Y=eye(2);
```

and finally, we can run the following script using Newton's method to obtain the right value of $u_1(0)$ (c in the script):

```
c=10;
aux=1;
while abs(c-aux)>1e-7
        aux=c;
        % call to sens_ind, to obtain the value of u1 at the end of the
        % time interval, yn(end,2), and the
        % derivative of this value with respect to u1(0): dydu(end,2,1);
        [tn,yn,dydu]=sens_sys('bvp',[0 1],[c0 ;0],[],[1;1],0,'ini_bvp');
        c0=c0-(yn(end,2))/(dydu(end,2,1));
end
plot(tn,yn)
```

2.5 Warnings and details to be completed

1) The main question which might be set is about the accuracy of the computed derivatives. It is difficult to give a precise answer, but it has been observed that SENS_SYS gives accuracies of the order of the integrator accuracies, while SENS_IND accuracies are typically worse. These accuracies can be controlled through the original ODE15s tolerances, as in the subsection 2.1; the tighter the ODE15s tolerances, the better the derivatives accuracy. Anyway, the accuracy attainable is problem-dependent.

It has been found that the default relative tolerance in ODE15s ($1e-3$) is too coarse to obtain good accuracies to the derivatives, so that it has been set to $1e-6$.

Both methods depend on the selection of a small increment *told*, used to build approximations to derivatives. The selection of this parameter is critical; at present, it has been used the same strategy as described in [3] , where *told* depends on the roundoff unit ε and on the size of the parameter u_i.

$$told_i = \sqrt{\varepsilon}\,(|u_i| + 0.1) \tag{5}$$

The 0.1 is added to avoid trouble when u_i is close (or equal) to zero. The strategy used in [6] is more sophisticated and robust, since it depends also solution of the system and the derivatives. However, this is simpler to implement and works well for reasonably scaled problems.

2) While some features of ODE15s have been properly extended, others are still not "corrected", such as the statistics collection, events or output functions.

3) Both subroutines have been tested with DAES of index 1 (the kind that ODE15s can solve) with good results; however, since the initial values for a DAE must be calculated, the derivatives of the initial conditions must be calculated as well. The approximation of these first order derivatives has been made through the standard first order expression (as below, in equation 7) which may be not too accurate. However, this should be a concern only for the initial value.

4) As has been mentioned, the vectorization of the m-files can make the subroutines SENS_IND and SENS_SYS fail when:
- Extra parameters are passed.
- The parameters (whose derivative needs to be found) passed have no influence on the system (as in the last example).

Finally, we want to remark that, since these subroutines are in development process, they should not be used for critical calculations, and their results must be checked carefully.

Next, a small sketch of the methods used is given, although we refer the interested reader to the original references.

3 Description of the methods

Both methods have been implemented trying to minimize the cost; the extra cost relative to that of ODE15s is due to some loops and an increase in function calls and triangular solves; no extra jacobians are computed or factorized.

3.1 Internal Numerical Differentiation[2]

The Internal Numerical Differentiation method just computes a set of "perturbed" trajectories, using exactly the same sequence of steps, orders, jacobian factorizations and solves than that of the "base trajectories", computed using the original unperturbed set of parameters. This is done by selecting an appropriate small increment for each parameter α_i (see equation 5).

The base system is integrated using the original parameter vector u and, using the same time steps sequence, the same jacobians, and, in general, the same sequence of operations, p integrations are performed with the parameter vectors:

$$u^{(i)} = u + \alpha_i e_i = u + \alpha_i (0, \cdots, \underbrace{1}_{i}, \cdots, 0)^t, \quad i = 1, \cdots, p \tag{6}$$

This will give a set of $(p+1)$ trajectories: the original y and the perturbed ones: $y^{(1)}, \cdots, y^{(p)}$. When all the trajectories have been computed, the derivatives are approximated with the known expression:

$$\frac{dy}{du_i} \approx \frac{y^{(i)} - y}{\alpha_i}. \tag{7}$$

Obviously, more accurate formulae may be used (the standard second order approximation) but it would need another p integrations of the system.

3.2 Iterative approximation with directional derivatives[6]

This approach has been described in several papers in terms of a more general parameter-dependent DAE system

$$F(t, y, y', u) = 0 \tag{8}$$

The sensitivity system is obtained differentiating the system (8)with respect to the parameter u, which shall give a new ODE/DAE system which may be solved along with the original.

The standard stepping algorithm (which is quite similar in ODE15s, DASSL, and others) may be applied directly to the sensitivity system, but this would not be too efficient. Instead, as proposed in [6], the sensitivity system may be approximated through a directional derivative finite difference approximation. Taking an small increment α_i for the parameter u_i (again, like in equation 5), and defining

$$s_i = \frac{dy}{du_i}, \tag{9}$$

the following systems may be set up (in terms of the general form (8))

$$\frac{F(t, y + \alpha_i s_i, y' + \alpha_i s_i', u + \alpha_i e_i) - F(t, y, y', u)}{\alpha_i} = 0, \quad i = 1, \cdots, p \tag{10}$$

where e_i is the ith unit vector. These p systems may be solved substituting y' and s_i' by their NDF (or BDF) approximations and solving the nonlinear algebraic system(which has the same jacobian of the original system) to obtain s_i.

References

[1] The Mathworks Inc., MATLAB 5.3 Natick MA (1998).

[2] Bock, H.G., Numerical Treatment of inverse problems in chemical reaction kinetics. In K.H. Ebert, P. Deuflhard and W. Jäger (eds.) *Modelling of Chemical Reaction Systems* (Springer Series in Chemical Physics 18). Springer, Heidelberg (1981).

[3] D. Leineweber: Analyse und Restrukturierung eines Verfahrens zur direkten Lösung von Optimal-Steuerungsproblemen (The Theory of MUSCOD in a Nutshell), http://www.iwr.uni-heidelberg.de/sfb/PP/Preprint1996-19.ps.gz

[4] L. F. Shampine and M. W. Reichelt, The Matlab ODE suite, *SIAM J. Sci. Comput.*, 18 pp. 1-22 (1997).

[5] L. F. Shampine, M. W. Reichelt and J.A. Kierzenka, Solving index-1 DAEs in MATLAB and Simulink, *Siam Review*, 41 pp. 538-552 (1999).

[6] T. Maly and L.R. Petzold, Numerical methods and software for sensitivity analysis of differential-algebraic systems, *Appl.Num. Math* 20 (1996) 57-79.

[7] Uri M. Ascher and Linda R. Petzold, Computer methods for Ordinary Differential equations and Differential-Algebraic equations, SIAM Publications, 1998.