

Manual SC-SR1

Johannes J. Brust, Oleg P. Burdakov, Jennnifer B. Erway and Roummel F. Marcia

The main function for solving unconstrained optimization problems is `LSR1_SC.m`

```
%% Inputs
% x0 (Initial iterate. Can be zeros(n,1) or ones(n,1))
% func (Objective function, f=func(x))
% grad (Objective gradient, g=grad(x))
% pars (Optional input struct, can be empty [],{} or
%       define one of several potential values, e.g., pars.print=1)

[xk1, gk1, fk1, out] = LSR1_SC( x0, func, grad, pars );

% out1 (Output struct, e.g., out1.ctime)
% fk1 (Objective value, fk1=func(xk1))
% gk1 (Gradient value, gk1=grad(xk1))
% xk1 (Solution estimate)
%% Outputs
```

To be able to use this function the folders `ALGS/` and `EXTERNAL/` need to be on the search path. Tables with most values of the (optional) input struct `pars` and output struct `out` are shown below. Additional descriptions of inputs and outputs are in the top lines of `LSR1_SC.m` (lines 2-99).

```
Struct pars, [Default]
    pars.tol := Tolerance; Stop if norm( $g_k$ , 'inf') < tol, [1e-6]
    pars.maxiter := Maximum iterations, [10000]
    pars.print := Flag to print iteration outputs, [0]
    pars.m := Memory parameter, [8]
    pars.SR1tol := SR1 update tolerance, [1e-8]
    pars.SR1tolAdj := Adjustment to SR1 update condition, [8]
    pars.c1 := TR acceptance. If pred/ared > c1 update  $x_k$ , [1e-5]
    pars.c2 := Good step. If pred/ared > c2 'good step', [0.75]
    pars.c3 := Closeness to boundary for 'good step', [0.8]
    pars.c4 := Increase radius by factor c4, [2]
    pars.c5 := Lower bound for 'ok step', [0.1]
    pars.c6 := Upper bound for 'ok step', [0.75]
    pars.c7 := Trust-region shrinkage for 'not good step', [0.5]
    pars.whichInit := "Quasi-Newton initialization  $B_0 = \gamma_k I$ ", [4]
    whichInit := 1 (all ones)
    whichInit := 2 (constant specified value)
    whichInitCons := 1 (Constant initialization within algorithm)
    whichInit := 3 ( $\gamma_k = y_k^T y_k / s_k^T y_k$  if positive, else no change)
    whichInit := 4 ( $\gamma_k = \max(y_k^T y_k / s_k^T y_k, \text{gams})$ , where gams stores "q" previous " $\gamma_k$ " values)
    pars.q := (If whichInit = 4), stores the previous "q" gamma values, [pars.m]
    pars.whichSub := "Which TR subproblem solver", [4]
    whichSub := 1 (sc_sr1_infty)
    whichSub := 2 (sc_sr1_2)
    whichSub := 3 (l2_sr1)
    whichSub := 4 (lstrs), additional parameters in pars.LSTRS
    whichSub := 5 (truncated CG), "additional parameters in pars.CG
    pars.initDelta := Initial trust-region radius, [1e5]
```

The default trust-region parameter values `pars.c1, ..., pars.c7` are related to [2, Algorithm 6.2]

Struct out

```
out.numiter := Number of iterations
out.numf := Number of function evaluations
out.numg := Number of gradient evaluations
out.ng := norm( $g_k$ , 'inf')
out.ex := Exit condition
out.Deltak := Trust-region radius
out.numAccept := Number step accepted
out.numTRInc := Number trust-region radius increase
out.numTRDec := Number trust-region radius decrease
out.ctime := Computational time
```

The subproblem algorithms are `sc_sr1_infty.m` and `sc_sr1_2.m`, which are called from within `LSR1_SC.m`. By setting the value of `pars.whichSub` a different subproblem solver can be selected.

```
%% Inputs
% g          (Gradient)
% S_or_Psi   (S or Psi = [S Y])
% Y_or_invM  (Y or invM)
% gamma_k    (B0 = gamma_k*I)
% delta      (Radius)
% flag       (=0 then S,Y. =1 then Psi,invM)
% show       (=0 w/o printing, =1 w. printing)
% varargin   (Variable inputs. Allows for potentially precomputed values,
%             Psi'*Psi (nargin=8) and/or invM (nargin=9))

[p_star_I, iExit_I] = sc_sr1_infty(g, S_or_Psi, Y_or_invM, gamma_k,...
    delta, flag, show);

[p_star_2, iExit_2] = sc_sr1_2(g, S_or_Psi, Y_or_invM, gamma_k,...
    delta, flag, show);

% p_star      (Subproblem solution estimate)
% iExit       (=0 converged, =1 error)
%% Outputs
```

Additional descriptions of the subproblem inputs are in the top lines of the two algorithms, i.e., lines 3-25.

1 Example 1

From within `EXPERIMENTS/` the script `example_1.m` is an illustration of applying `LSR1_SC` to the Rosenbrock objective function. The objective function and the gradient are defined in `AUXILIARY/rosen_obj.m` and `AUXILIARY/rosen_grad.m`. Selected commands from `example_1.m` and outputs are below

```
% Additions of folders to the path
addpath(genpath('..'/ALGS)); addpath(genpath('..'/EXTERNAL));
addpath(genpath('..'/AUXILIARY));

% Rosenbrock objective function and gradient
% Inputs to LSR1_SC
func = @(x)( rosen_obj(x) );
grad = @(x)( rosen_grad(x) );

% Problem dimension
n = 10000;

% This example uses default parameters, which are described in
% the comments of LSR1_SC.m. Only exceptions are to print the outputs
% and to use an identity initialization
```

```

pars.print = 1;
pars.whichInit = 1;

% Initial guess (solution is xsol=ones(n,1), f(xsol)=0)
x0 = zeros(n,1);
x0(1) = 30;

% Call to the optimization algorithm
[xk1, gk1, fk1, out1] = LSR1_SC(x0, func, grad, pars);

```

Outputs of running this example show that the algorithm converged to default tolerances in 34 iterations.

Example 1 #####

Rosenbrock objective: $f(x)$

```

        n = 10000
        f0 = 8.1584e+05
        norm(g0) = 1.0807e+05

```

Algorithm: SC_SR1

#####

```

----- Running Algorithm -----

```

Iter	fk	norm(gk)	TR	numf
0	8.1584e+05	1.0806e+05	1.0000e+05	1
1	5.0087e+03	2.5728e+01	6.3640e+01	7
2	3.3360e+03	4.8177e+00	6.3640e+01	8
3	1.1871e+03	1.0034e+01	6.3640e+01	9
4	2.7365e+02	3.3781e+01	6.3640e+01	10
5	2.7365e+02	3.3781e+01	3.1820e+01	11
6	2.9117e+01	1.4049e+01	3.1820e+01	12
7	8.9110e+00	1.3949e+00	3.1820e+01	13
8	8.3665e+00	2.5228e+00	3.1820e+01	14
9	7.5493e+00	5.1597e+00	3.1820e+01	15
10	5.8645e+00	1.9872e+00	3.1820e+01	16
11	5.8645e+00	1.9872e+00	1.5910e+01	17
12	5.8645e+00	1.9872e+00	7.9551e+00	18
13	5.4004e+00	3.4931e+00	7.9551e+00	19
14	3.5463e+00	4.4320e+00	7.9551e+00	20
15	3.5463e+00	4.4320e+00	3.9775e+00	21
16	2.5512e+00	2.7267e+00	3.9775e+00	22
17	8.0685e-01	1.2023e+00	3.9775e+00	23
18	5.5232e-01	9.5339e-01	3.9775e+00	24
19	5.5232e-01	9.5339e-01	1.9888e+00	25
20	5.5232e-01	9.5339e-01	9.9438e-01	26
21	5.5232e-01	9.5339e-01	4.9719e-01	27
22	5.5232e-01	9.5339e-01	2.4860e-01	28
23	2.7307e-01	4.8238e-01	4.9719e-01	29
24	2.6827e-01	1.8369e+00	2.4860e-01	30
25	3.2366e-02	2.1208e-01	4.9719e-01	31
26	2.8302e-02	1.8933e-01	4.9719e-01	32
27	2.8302e-02	1.8933e-01	2.4860e-01	33
28	7.0826e-04	6.8205e-02	4.9719e-01	34
29	2.8262e-04	3.6591e-02	4.9719e-01	35
30	1.4621e-06	3.5354e-03	4.9719e-01	36
31	1.7701e-07	6.7013e-04	4.9719e-01	37
32	1.7701e-07	6.7013e-04	2.4860e-01	38
33	1.0780e-11	1.0807e-05	2.4860e-01	39
34	6.1319e-17	1.9119e-08	2.4860e-01	40

>>

2 Example 2

The example `EXPERIMENTS/example_2.m` illustrates three different subproblem solvers. Likewise, some of the optional parameters are set to non-default values. For instance, the convergence tolerance is `pars.tol = 10-4`, the maximum iterations `pars.maxiter = 200`.

```
addpath(genpath('..../ALGS'));
addpath(genpath('..../EXTERNAL'));
addpath(genpath('..../AUXILIARY'));

% Rosenbrock objective function and gradient
func = @(x)( rosen_obj(x) );
grad = @(x)( rosen_grad(x) );

% Problem dimensions
ns = [500, 1000, 5000, 10000, 50000, 100000, 300000];

% Setting some trust-region algorithm parameters
pars.tol    = 1e-4;
pars.print  = 0;
pars.maxiter= 200;
pars.m      = 5;

% Loop over problem dimensions
for i = 1:length(ns)

    % Initial point
    n      = ns(i);
    x0     = zeros(n,1);
    x0(1)  = 30;

    % Using sc_sr1_infty.m (Shape-changing infinity norm solver, Alg. 3)
    pars.whichSub    = 1;
    [xk1,gk1,fk1,out1] = LSR1_SC(x0,func,grad,pars);

    % Using sc_sr1_2.m (Shape-changing 2 norm solver, Alg. 4)
    pars.whichSub    = 2;
    [xk2,gk2,fk2,out2] = LSR1_SC(x0,func,grad,pars);

    % Using obs.m (L2 norm solver)
    pars.whichSub    = 3;
    [xk3,gk3,fk3,out3] = LSR1_SC(x0,func,grad,pars);

    fprintf('i \t %3.1e  %3.1e \t %3.1e  %3.1e \t %3.1e  %3.1e \n',n,...
            out1.ctime,out1.ng,out2.ctime,out2.ng,out3.ctime,out3.ng);

end
```

Outcomes from running the three subproblem solvers on the Rosenbrock objective with 7 different dimensions is shown below. All algorithms are converging to the desired tolerance.

Example 2 #####

Rosenbrock objective: f(x)

n = [500, 1000, 5000, 10000, 50000, 100000, 300000]

Sub. Algorithms: TR:SC-INF, TR:SC-L2, TR:L2

#####

n	TR:SC-INF		TR:SC-L2		TR:L2	
-	Time	norm(g)	Time	norm(g)	Time	norm(g)
500	9.5e-03	7.1e-05	8.9e-03	5.0e-05	7.2e-03	9.7e-05
1000	7.8e-03	6.6e-05	1.2e-02	7.3e-05	1.1e-02	8.6e-06
5000	3.0e-02	2.6e-07	2.6e-02	1.8e-05	2.3e-02	8.6e-07
10000	5.5e-02	3.2e-09	5.4e-02	4.6e-05	3.9e-02	1.7e-05
50000	3.8e-01	3.2e-05	3.3e-01	1.2e-06	3.5e-01	2.3e-06
100000	5.8e-01	9.2e-05	7.6e-01	4.4e-05	6.7e-01	1.7e-05
300000	1.4e+00	5.1e-06	1.8e+00	4.2e-05	2.0e+00	2.2e-07

>>

References

- [1] J.J. Brust, O.P. Burdakov, J.B. Erway and R.F. Marcia Algorithm xxx: SC-SR1: MATLAB Software for Limited-Memory SR1 Trust-Region Methods. *ACM Trans Math Softw* (2022)
- [2] J. Nocedal and S.J. Wright, *Numerical Optimization*. Springer, New York (2006)