The RCDS Algorithm and Its Matlab Implementation for Automated Tuning

X. Huang, SLAC, 2575 Sand Hill Road, Menlo Park, CA 94025
xiahuang@slac.stanford.edu
(6/24/2013)

The RCDS (robust conjugate direction method) is an optimization method for multi-variable, noisy objective functions. This method performs iterative line scans to minimize a function (single objective) over a set of directions. These directions are preferred to be mutually conjugate. The main advantage of the method is its capability to find the optimum in a noisy environment, which makes it suitable for automated tuning. This capability is achieved through the robust line optimizer. Please refer to Ref. [1] for more details.

There are risks when using an automated algorithm to run a machine. The RCDS algorithm provides a mechanism for the user to limit and check the ranges of the parameters. However, because of the complexity of the potential applications, it is impossible to rule out the possibility that something goes wrong. The responsibility of machine or personnel safety lies entirely with the user of the algorithm/code.

I. Description of the algorithm and the package

The RCDS algorithm has two components: Powell's algorithm to update the conjugate direction set and the robust line optimizer to look for the minimum on each direction. The implementation has three functions:

- powellmain.m: This is the backbone of the algorithm. It implements Powell's method as described in Numerical Recipe [2]. It contains the main iteration loop. For each iteration it searches all directions, one at a time. At the end of the iteration the direction with the largest decrease of the objective may be replaced with the direction from the best solution of the previous iteration to the present best solution. For the search in each direction, the bounds that enclose a minimum is first established with the function "bracketmin.m". Then a line scan is performed within the bounded region with the function "linescan.m".
- bracketmin.m: Given a function handle, an initial solution, a line in the parameter space, and an initial step size, this function goes both directions on the line to find two points whose objective function values are significantly higher (i.e., by more than three times the noise sigma) than that of a point in between. If the boundary of the parameter space is met before a true bound is found, the last evaluated point is used as the bound in that direction.
- linescan.m: This function searches the bounded region as determined by "bracketmin.m" to find the minimum within. The solutions evaluated in "bracketmin.m" are passed to this function and reused. A few new solutions along the line are inserted to fill the large distance between the existing solutions. A quadratic polynomial fitting is performed for all solutions in the bounded region. Outliers in the data points are detected with the function "outlier1d". If there is one outelier, it is removed and data are refitted. If there

are two or more outliers, the fitting is discarded and the evaluated solution with the minimum objective is returned. Otherwise evaluate 101 points on the fitted parabola within the bounded region and find the minimum.

II. Usage of the Matlab RCDS package

The code for the algorithm is written to be generally applicable. The three Matlab functions described above are not problem-specific. When we apply the algorithm to a new problem, we only need to provide an objective function and launch the algorithm. We illustrate the use of the algorithm with an example below. Please look into the functions “func_obj”, “run_RCDS_test” under the sub-directory “ex0” for details. The four functions in the previous section need to be in the Matlab path to run this example. Try the example in Matlab with

```
>> run_RCDS_test
```

Defining the objective function

For this example the objective function is defined in the function “func_obj”. The input of this function is the parameter vector, with each variable normalized to the zone [0, 1]. The original ranges of the variables are defined in the global variable “vrange”. The first block of the function checks the values of the parameters. If any parameter falls outside of [0, 1], the function returns NaN (Not a Number). Otherwise, the real parameter vector, “p”, is calculated using “vrange” and “x”.

In the second block we evaluate the objective function. For a real online problem, we would change the machine setpoints and measure the value of the objective here. In the example we simply use a formula to calculate the objective function and add random noise to it.

The parameter vector and the objective function are then appended to a global variable - the “g_data” matrix. In this way we save the data in the Matlab global space, which will not be lost even if the program crashes or is terminated manually (which is usually the case). You may also want to print out the solutions to the Matlab window to monitor the optimization process.

Launching RCDS

Setting up and launching the algorithm is done in the script “run_RCDS_test.m”. In this script we define and initialize the global variables, which are

Nvar – the number of parameters.
vrange – the range of parameters
g_cnt – the number of evaluated solutions
g_data – data associated with all the evaluated solutions
g_noise – the noise sigma of the objective function

The noise level is an important parameter required by the algorithm. It can be evaluated by taking many evaluations of the same solution and compute the standard deviation.

The algorithm is launched by calling the function “powellmain”.

```
[x1,f1,nf]=powellmain(@func_obj,x0,step,dmat,[],100,'noplot',2000)
```

where the input arguments are

@func_obj – the function handle to the objective function.

x0 – the initial parameter vector (normalized).
step – the initial step size for bracketing. It is reasonable to set step=0.01.
dmat – the matrix that contains the directions (each column vector is a direction). The number of directions should be the same as the number of variables.
tol – tolerance, which can be used to define a termination condition. You can disable this feature by setting tol to 0 or a negative number.
maxIt – maximum number of iterations, set to 100 in this example
flag_plot – a flag indicating if we want to plot the bracketing and line scan data.
maxEval – the maximum of function evaluations. This is often used as a termination condition.

Terminating, examining data and applying the best solution

The algorithm may be terminated using the conditions defined with "tol" or "maxEval". But more frequently we may simply terminate the running code with "Ctrl+C" after a satisfactory gain has been made or it is no longer making appreciable gains or the shift is running out of time. A forced termination will not lose data since they are saved in the global variable "g_data". However, we need to save the data to a file immediately and make sure the file name differs from previously saved files to avoid overwriting. It is advisable to back up the data file immediately after it is saved.

The function "process_data" can be used to examine and plot the data. It calculates the normalized values of the parameters of all solutions, calculates the history of the best solution and calculates the distance of each solution to the preferred solution in normalized coordinates. The preferred solution can be the solution with the minimum objective function ('best" solution), or a solution that is next to the best if we suspect the best solution is an outlier. This is specified with the "isel" argument to the "process_data" function.

If we choose to plot results, "process_data" plots the objectives, the distances to the preferred solution, and the normalized parameters of all solutions.

The "process_data" function returns the preferred solution in normalized parameters which can be applied to the machine directly with the objective function "func_obj".

If the optimization algorithm is otherwise interrupted (e.g. crashed), we also save data as described above. We may also want to apply the best solution. After that the code can start again automatically from the best solution if we use the present machine setpoints as the initial solution (this can be set in run_RCDS_test.m).

The initial conjugate direction set

In theory Powell's method is able to develop a conjugate direction set gradually from search results. However, the limited machine study time often means the code doesn't have time to replace and refine the initial direction set. For some problems it is very important to provide a good initial direction set.

We can always use the identity matrix as the initial direction set. This is also the default direction set. But ideally we want to use a model (even if the model is very rough) to figure out a conjugate direction set. If we can evaluate the Hessian matrix **H** of the objective function with respect to the parameters, then its eigenvectors are a good set of initial directions.

In some cases we can identify a set of conditions that are equivalent or nearly equivalent to the optimal condition. Calculating the derivatives of the parameters in these conditions with respect to the knob parameters (the Jacobian) may be more convenient than calculating the second order derivatives directly as required for the Hessian. For example, for the coupling correction problem, the condition of minimum coupling is nearly equivalent to small off-diagonal elements on the orbit response matrix. We can calculate the Jacobian matrix **J** of these off-diagonal elements. A direction set is then given by the singular value decomposition (i.e., the V-matrix of $[\mathbf{U},\mathbf{S},\mathbf{V}] = \mathrm{svd}(\mathbf{J})$, this is the case since $\mathbf{H}=\mathbf{J}^T\mathbf{J}$).

References:

1. X. Huang, J. Corbett, J. Safranek, J. Wu, Nucl. Instr. Methods, A, 726 (2013) 77-83.
2. W. H. Press, et al, Numeric Recipies, Cambridge University Press (2007).