Release-Notes to `direct.m`
June 22, 2004

This short document provides a description of the updates to the `direct.m` software. The software, this document, and several examples can be found at

```
http://www4.ncsu.edu/~definkel/research/index.html
```

The examples provided on the web-site are meant to supplement this document, and clear up any ambiguities about how to use the new features of `direct.m`.

The code `direct.m` is now designed to solve problems of the form

$$(P)\qquad \begin{array}{ll} \min & f(x) \\ \text{subject to} & c(x) \leq 0, \quad l \leq x \leq u. \end{array}$$

The ability of the code to address constraints is a new feature. The software still uses the `DIRECT` algorithm to solve problems. When a point is sampled that violates the constraints, it assigns an artificial function value to that point. There are two approaches to determining the artificial value, and we describe them both below.

Change in Calling Structure: There is a subtle change to the calling structure of `direct.m`. In older versions, the objective function was passed to the program as a character array. In the new version, it is passed as part of a strucutre. The calling sequence is still of the form

```
>> [fmin,xmin,history] = Direct(Problem,bounds,options);
```

Here, `Problem` is a MATLAB structure. For a simple, bound-constrained problem, you only need to set one field of the `Problem` structure

```
>> Problem.f = 'myTestProblem';
```

where your objective function is the MATLAB file `myTestProblem.m`. The variable `bounds` is still an $n \times 2$ vector of the lower and upper bounds for each variable, and `options` is a MATLAB structure which contains specific options for the program. The options structure is described in the sample files, and by typing

```
>> help Direct
```

at the command prompt.

Choices for handling additional constraints The new version of `direct.m` has two methods built into it for solving problems which have constraints in addition to those on the bounds. Note, if you have the output turned on, then an asterisk will be shown next to infeasible function values.

- $L1$ Penalty Functions.
  One approach is to transform the constrained problem into an unconstrained one. If we are trying to solve Problem (P), then we can instead try to solve

  $$\min f(x) + \mu^T \max\left(c(x), 0\right)$$

  where $l \leq x \leq u$, and μ is a user-supplied penalty parameter. This method can be very effective because information about the feasibility of a point is included in the objective function. It does, however, require explicit knowledge of the constraint, and an educated guess at the appropriate value for the penalty parameter. For example, say we are trying to solve

  $$\begin{array}{lll} \min & f(x,y) = x + y & \\ \text{subject to} & c_1(x,y) = x^2 + y^2 - 6 & \leq \;\; 0 \\ & c_2(x,y) = 2x + y & \leq \;\; 0 \end{array}$$

  and $-10 \leq x, y \leq 10$. We assume that the objective function, and the two constraints are in three separate files named `objfcn.m`, `c1.m` and `c2.m`, respectively. The following shows a sample call to `direct.m` for this problem:

  ```
  >> bounds = [-10 10;-10 10];
  >> Problem.f = 'objfcn';
  >> Problem.constraint(1).func = 'c1';
  >> Problem.constraint(1).penalty = 1;
  >> Problem.constraint(2).func = 'c2';
  >> Problem.constraint(2).penalty = 1;
  >> Problem.numconstraints = 2;
  >> options. ... (set your options)
  >> [fmin,xmin,hist] = Direct(Problem,bounds,options);
  ```

  For each constraint, you are allowed to choose a unique penalty parameter for it. NOTE, if you do not wish to use penalty function capabilities, simply leave the `Problem.constraint`, and `Problem.numconstraints` fields empty.

- Neighborhood Assignment Strategy (NAS)
  This approach was first described in [2], and was suggested by R. Carter. We do not go into the details, and instead refer the reader to [2, 1] for more information on the details of this approach.

If the constraints $c(x)$ are not explicitly known, or you do not wish to use penalty approach for addressing your c, `direct.m` has a feature which allows the algorithm to assign a value to infeasible points based on the values of feasible points in it's neighborhood.
To use this approach, add the additional option:

```
>> options.impcons = 1;
```

By setting this option, `direct.m` will now expect the objective function to return two values. A sample header for the objective function is now

```
 [retval, fflag] = testfcn(x);
```

The variable `fflag` is set to zero if the point sampled is feasible, and one otherwise. This lets `direct.m` know if it needs to calculate and assign an artificial value for this point.
A sample call to the program is:

```
>> bounds = [-10 10;-10 10];
>> Problem.f = 'objfcn';
>> options. ... %set your options
>> options.impcons = 1;
>> [fmin,xmin,hist] = Direct(Problem,bounds,options);
```

Please see the sample files on the web-site for complete examples.

References

[1] J. M. Gablonsky. DIRECT Version 2.0 User Guide. Technical Report CRSC-TR01-08, Center for Research in Scientific Computation, North Carolina State University, April 2001.

[2] J.M. Gablonsky. *Modifications of the DIRECT Algorithm.* PhD thesis, North Carolina State University, 2001.