Utilities

The Hooke-Jeeves Function (TOMS178)

Reference:

http://people.sc.fsu.edu/~jburkardt/m_src/toms178/toms178.html

Introduction

This worksheet implements the Hooke-Jeeves search algorithm. It is a port into Mathcad of Dr John Burkardt's Matlab code given in the Reference.

Discussion

function [iters, endpt] = hooke (startpt, ρ, ε, itermax, f)

hooke seeks a minimizer of a scalar function of several variables.

This routine find a point X where the nonlinear objective function
F(X) has a local minimum. X is an N-vector and F(X) is a scalar.

The objective function F(X) is not required to be differentiable or even continuous. The program does not use or require derivatives of the objective function.

The user supplies three things:

1. a subroutine that computes F(X),
2. an initial "starting guess" of the minimum point X,
3. values for the algorithm convergence parameters.

The program searches for a local minimum, beginning from the starting guess, using the Direct Search algorithm of Hooke and Jeeves.

This program is adapted from the Algol pseudocode found in the paper by Kaupe, and includes improvements suggested by Bell and Pike, and by Tomlin and Smith.

The algorithm works by taking "steps" from one estimate of a minimum, to another (hopefully better) estimate. Taking big steps gets to the minimum more quickly, at the risk of "stepping right over" an excellent point. The stepsize is controlled by a user supplied parameter called ρ. At each iteration, the stepsize is multiplied by ρ ($0 < \rho < 1$), so the stepsize is successively reduced.

Small values of ρ correspond to big stepsize changes, which make the algorithm run more quickly. However, there is a chance (especially with highly nonlinear functions) that these big changes will accidentally overlook a promising search vector, leading to nonconvergence.

Large values of ρ correspond to small stepsize changes, which force the algorithm to carefully examine nearby points instead of optimistically forging ahead. This improves the probability of convergence.

The stepsize is reduced until it is equal to (or smaller than) ε. So the number of iterations performed by Hooke-Jeeves is determined by ρ and ε:

$$\rho^{number\_of\_iterations} = \varepsilon$$

In general it is a good idea to set ρ to an aggressively small value like 0.5 (hoping for fast convergence). Then, if the user suspects that the reported minimum is incorrect (or perhaps not accurate enough), the program can be run again with a larger value of ρ such as 0.85, using the result of the first minimization as the starting guess to begin the second minimization.

Normal use:
Code your function F() in Mathcad

Install your starting guess;
Run the program.

If there are doubts about the result, the computed minimizer can be used as the starting point for a second minimization attempt.

To apply this method to data fitting, code your function F() to be the sum of the squares of the errors (differences) between the computed values and the measured values. Then minimize F() using Hooke-Jeeves.

For example, you have 20 datapoints (t_i, y_i) and you want to find a, b and c so that:

```
a*t*t + b*exp(t) + c*tan(t)
```

fits the data as closely as possible. Then the objective function F() to be minimized is just

```
F(a,b,c) = sum ( 1 <= i <= 20 )
  ( y(i) - a*t(i)*t(i) - b*exp(t(i)) - c*tan(t(i)) )^2.
```

or, in Mathcad form:

$$F(a,b,c) = \sum_{i=0}^{last(t)} \left[y_i - \left[a \cdot \left(t_i\right)^2 + b \cdot \exp\left(t_i\right) - c \cdot \tan\left(t_i\right) \right] \right]^2$$

Modified:
12 Jul 12

Author:
ALGOL original by Arthur Kaupe.
C version by Mark Johnson.
MATLAB version by John Burkardt.
Mathcad version by Stuart Bruff

References:

M Bell, Malcolm Pike,
Remark on Algorithm 178: Direct Search,
Communications of the ACM,
Volume 9, Number 9, September 1966, page 684.

Robert Hooke, Terry Jeeves,
Direct Search Solution of Numerical and Statistical Problems,
Journal of the ACM,
Volume 8, Number 2, April 1961, pages 212-229.

Arthur Kaupe,
Algorithm 178:
Direct Search,
Communications of the ACM,
Volume 6, Number 6, June 1963, page 313.

FK Tomlin, LB Smith,
Remark on Algorithm 178: Direct Search,
Communications of the ACM,
Volume 12, Number 11, November 1969, page 637-638.

Arguments:

Input, real STARTPT(NVARS), the user-supplied initial estimate for the minimizer.
Input, real ρ, a user-supplied convergence parameter which should be set to a value between 0.0 and 1.0. Larger values of ρ give greater probability of convergence on highly nonlinear functions, at a cost of more function evaluations. Smaller values of ρ reduce the number of evaluations and the program running time, but increases the risk of nonconvergence.
Input, real ε, the criterion for halting the search for a minimum. When the algorithm begins to make less and less progress on each iteration, it checks the halting criterion: if the stepsize is below ε, terminate the iteratio and return the current best estimate of the minimum. Larger values of ε (such as 1.0e-4) give quicker running time, but a less accurate estimate of the minimum. Smaller values of ε (such as 1.0e-7) give longer running time, but a more accurate estimate of the minimum.
Input, integer ITERMAX, a limit on the number of iterations.
Input, function handle F, the name of the function routine, which should have the form:
function value = f (x, n)

Output, integer ITERS, the number of iterations taken.
Output, real ENDPT(NVARS), the estimate for the minimizer, as calculated by the program.

where NVARS is the number of dimensions

The original Matlab implementation is contained within the Area "Hooke-Jeeves Matlab implementation" below A version of the Mathcad code is also given that uses nvars as an input argument. The main Mathcad implementation doesn't use nvars as an argument, but rather determines it from the size of the appropriate input arguments. An important point to note is that the default Mathcad array index is zero (0) rather than Matlab's one (1) (although this could be changed by setting the built-in variable ORIGIN to 1).

Hooke-Jeeves Matlab Implementation

Mathcad Implementation

Implementation Notes

The Mathcad functions below are structurally close to their Matlab equivalents and primarily use built-in Mathcad functions for their implementation. Exceptions to this latter rule are the use of the user-defined function subvector (used because it makes the functions more readable), vec, which is of particular use in converting scalar and nested arrays to vectors in the Examples section and linspace, which is a Mathcad implementation of the Matlab function of the same name.

Hooke-Jeeves Function

```
best_nearby(Δ, point, prevbest, f, funevals) :=
    nvars ← rows(point) − 1
    z ← subvector(point, 0, nvars)
    minf ← prevbest
    for i ∈ 0 .. nvars
        z_i ← point_i + Δ_i
        ftmp ← f(z)
        funevals ← funevals + 1
        minf ← ftmp if ftmp < minf
        otherwise
            Δ_i ← −Δ_i
            z_i ← point_i + Δ_i
            ftmp ← f(z)
            funevals ← funevals + 1
            minf ← ftmp if ftmp < minf
            z_i ← point_i otherwise
    point ← subvector(z, 0, nvars)
    (minf  point  funevals)
```

```
hooke(startpt, ρ, ε, itermax, f) :=
    nvars ← rows(startpt) − 1
    newx ← subvector(startpt, 0, nvars)
    xbefore ← newx
    for i ∈ 0 .. nvars
        Δ_i ← ρ·if(startpt_i, |startpt_i|, 1)
    funevals ← 0
    steplength ← ρ
    iters ← 0
    fbefore ← f(newx)
    funevals ← funevals + 1
    newf ← fbefore
    while iters < itermax ∧ ε < steplength
        iters ← iters + 1
        newx ← subvector(xbefore, 0, nvars)
        (newf  newx  funevals) ← best_nearby(Δ, newx, fbefore, f, funevals)
        keep ← 1
        while newf < fbefore ∧ keep = 1
            for i ∈ 0 .. nvars
                Δ_i ← |Δ_i|·if(newx_i ≤ xbefore_i, −1, 1)
                tmp ← xbefore_i
                xbefore_i ← newx_i
                newx_i ← newx_i + newx_i − tmp
            fbefore ← newf
            (newf  newx  funevals) ← best_nearby(Δ, newx, fbefore, f, funevals)
```

```
        break  if  fbefore ≤ newf
        keep ← 0
        for  i ∈ 0 .. nvars
          if  0.5·|Δ_i| < |newx_i − xbefore_i|
            keep ← 1
            break
      if  ε ≤ steplength ∧ fbefore ≤ newf
        steplength ← steplength·ρ
        Δ ← Δ·ρ
    (iters  subvector(xbefore, 0, nvars))
```

General Purpose Hooke-Jeeves Wrapper

$$\text{hookejeeves}(f, \text{startpt}) := \left| \begin{array}{l} \text{startpt} \leftarrow \text{vec}(\text{startpt}) \quad \text{if} \quad \text{rows}(\text{startpt}) = 0 \\ \text{itermax} \leftarrow 5000 \\ \rho \leftarrow 0.5 \\ \varepsilon \leftarrow 10^{-6} \\ (\text{iters} \quad \text{endpt}) \leftarrow \text{hooke}(\text{startpt}, \rho, \varepsilon, \text{itermax}, f) \\ (f(\text{startpt}) \quad \text{iters} \quad \text{endpt} \quad f(\text{endpt})) \end{array} \right.$$

Examples

Rosenbrock Function

$$\text{rosenbrock}(\mathbf{x}) := 100 \cdot \left[\mathbf{x}_1 - (\mathbf{x}_0)^2\right]^2 + (1 - \mathbf{x}_0)^2$$

$$\text{rosenbrock}(\text{stack}(-1.2, 1.0)) = 24.2$$

$$\text{hookejeeves}(\text{rosenbrock}, \text{stack}(-1.2, 1.0)) = \begin{bmatrix} 24.2 & 19 & \begin{pmatrix} 1.000001 \\ 1.000002 \end{pmatrix} & 1.513395 \times 10^{-11} \end{bmatrix}$$

Matlab results

TEST01
Here we use the Rosenbrock function.
Initial estimate X =
1 -1.200000e+00
2 1.000000e+00
F(X) = 2.420000e+01

Number of iterations taken = 19
X* =
1 1.000001e+00
2 1.000002e+00
F(X*) = 1.513395e-11

Dennis-Woods Function

$$\begin{aligned}
\text{woods}(\mathbf{x}) := \;& s1 \leftarrow \mathbf{x}_1 - (\mathbf{x}_0)^2 \\
& s2 \leftarrow 1 - \mathbf{x}_0 \\
& s3 \leftarrow \mathbf{x}_1 - 1 \\
& t1 \leftarrow \mathbf{x}_3 - (\mathbf{x}_2)^2 \\
& t2 \leftarrow 1 - \mathbf{x}_2 \\
& t3 \leftarrow \mathbf{x}_3 - 1 \\
& t4 \leftarrow s3 + t3 \\
& t5 \leftarrow s3 - t3 \\
& value \leftarrow 100 \cdot s1^2 \ldots \\
& \qquad + s2^2 \ldots \\
& \qquad + 90 \cdot t1^2 \ldots \\
& \qquad + t2^2 \ldots \\
& \qquad + 10 \cdot t4^2 \ldots \\
& \qquad + 0.1 \cdot t5^2 \\
& value
\end{aligned}$$

$$\text{woods}(\text{stack}(-3,-1,-3,-1)) = 1.9192 \times 10^4$$

$$\text{hookejeeves}(\text{woods}, \text{stack}(-3,-1,-3,-1)) = \begin{bmatrix} 1.9192 \times 10^4 & 19 & \begin{pmatrix} 1.000134 \\ 1.000269 \\ 0.999865 \\ 0.999729 \end{pmatrix} & 6.56948 \times 10^{-8} \end{bmatrix}$$

Matlab results

TEST02
Here we use the Woods function.
Initial estimate X =
1 -3.000000e+00
2 -1.000000e+00
3 -3.000000e+00
4 -1.000000e+00
F(X) = 1.919200e+04

Number of iterations taken = 19
X* =
1 1.000134e+00
2 1.000269e+00
3 9.998646e-01
4 9.997292e-01
F(X*) = 6.569480e-08

Some Other Function ...

This is quite a familiar looking function. I've lazily dealt with maximization by negating the function; this negates the result, but that can be taken care of when using the result. The 3D plot below shows the surface and max and min local peaks starting from some arbitrary point. The 2 peaks are shown as large points connected by lines.

$$\text{someFunction}(v) := \left| \begin{array}{l} (x \quad y) \leftarrow v^{T} \\ 3\cdot(1-x)^{2}\cdot\exp\left[-x^{2}-(y+1)^{2}\right] \ldots \\ +-10\cdot\left(\frac{1}{5}x - x^{3} - y^{5}\right)\cdot\exp\left(-x^{2}-y^{2}\right) \ldots \\ +-\frac{1}{3}\exp\left[-(x+1)^{2}-y^{2}\right] + 0.1\cdot\left(x^{2}+y^{2}\right) \end{array} \right.$$

$$\text{someOtherFunction}(x) := -\text{someFunction}(x)$$

$$x0 := \text{stack}(2,1)$$

$$\text{someFunction}(x0) = 1.08$$

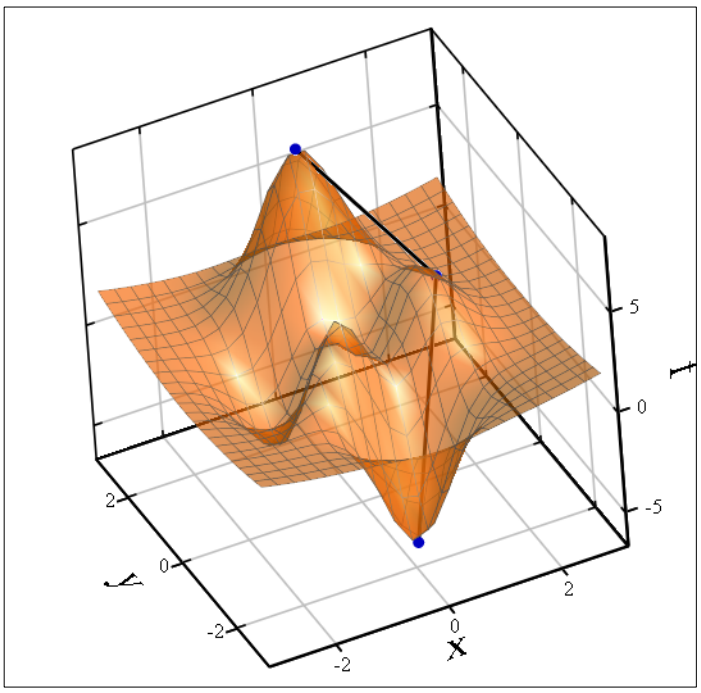
$$\text{hjmin} := \text{hookejeeves}(\text{someFunction}, x0) = \left[1.080455 \quad 19 \quad \begin{pmatrix} 0.22813 \\ -1.614429 \end{pmatrix} \quad -6.283497\right]$$

$$\text{hjmax} := \text{hookejeeves}(\text{someOtherFunction}, x0) = \left[-1.080455 \quad 19 \quad \begin{pmatrix} -9.1362\times10^{-3} \\ 1.591221 \end{pmatrix} \quad -8.357851\right]$$

$$\text{hjminmax} := \left[\left[\begin{pmatrix} (\text{hjmin}_{0,2})_{0} \\ x0_{0} \\ (\text{hjmax}_{0,2})_{0} \end{pmatrix}\right] \left[\begin{pmatrix} (\text{hjmin}_{0,2})_{1} \\ x0_{1} \\ (\text{hjmax}_{0,2})_{1} \end{pmatrix}\right] \begin{pmatrix} \text{hjmin}_{0,3} \\ \text{someFunction}(x0) \\ -\text{hjmax}_{0,3} \end{pmatrix}\right]^{T}$$

$$\text{hjminmax}^{T} = \left[\begin{pmatrix} 0.228 \\ 2 \\ -9.136\times10^{-3} \end{pmatrix} \begin{pmatrix} -1.614 \\ 1 \\ 1.591 \end{pmatrix} \begin{pmatrix} -6.283 \\ 1.08 \\ 8.358 \end{pmatrix}\right]$$

$$f(x,y) := \text{someFunction}(\text{stack}(x,y))$$



f , hjminmax

Optimizing scalar functions

The implementation requires that the input argument to a function be a vector; this, unfortunately, makes the arguments a little cumbersome when dealing with a scalar function. We can deal with this in several ways: one, and possibly the best from a user perspective, is to rewrite the hooke functions to handle scalar argument functions (say, by choosing between a scalar and a vector version); another is to redefine the function to vectorize the input argument. The version below takes the latter approach, and uses a Math Style to allow typographic distinction between the scalar and vector forms (*in addition, hookejeeves, partially accomodates a scalar argument by converting it to a vector*).

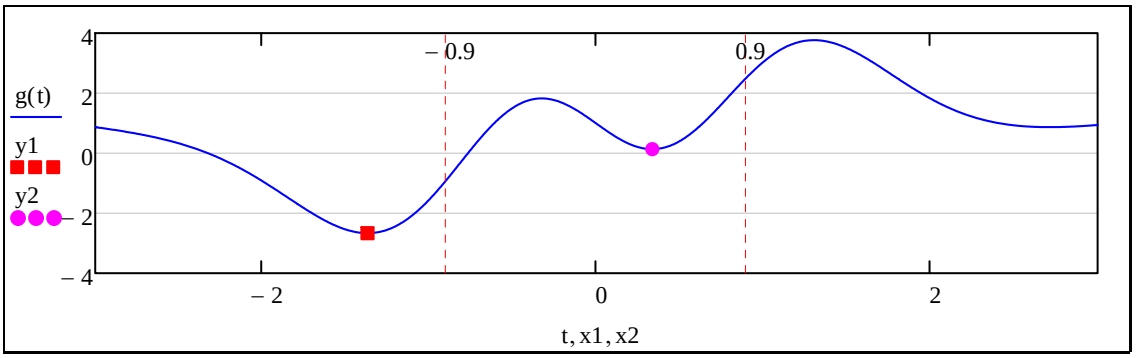
We'll create a specialized form of someFunction that takes the section through y = 0 and look for the minima starting from x = -0.9 and x = 0.9.

$$g(x) := 3\cdot(1-x)^2\cdot\exp\left(-x^2-1\right) \ldots + -10\cdot\left(\frac{1}{5}x - x^3\right)\cdot\exp\left(-x^2\right) \ldots + -\frac{1}{3}\exp\left[-(x+1)^2\right] + 0.1\cdot\left(x^2\right)$$

$$\mathbf{g}(t) := g\left(t_0\right)$$

$m1 := \text{hookejeeves}(\mathbf{g}, -0.9)$ $\quad m1 = [-0.919 \quad 19 \quad (-1.369) \quad -2.671]$ $\quad x1 := \left(m1_{0,2}\right)_0$ $\quad y1 := m1_{0,3}$

$m2 := \text{hookejeeves}(\mathbf{g}, 0.9)$ $\quad m2 = [2.519 \quad 20 \quad (0.334) \quad 0.129]$ $\quad x2 := \left(m2_{0,2}\right)_0$ $\quad y2 := m2_{0,3}$

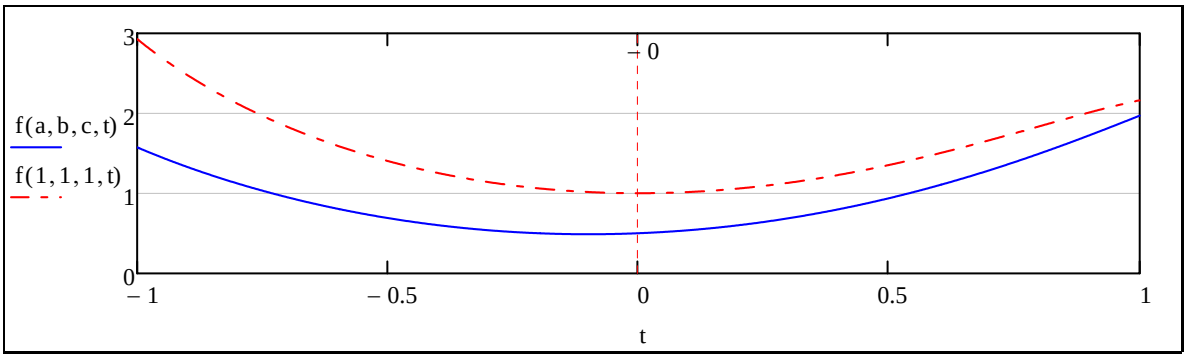


Fitting data

The example below uses the fitted data outline example given in the Discussion section above. We'll define and plot the associated function first, together with a set of 'starting point' values for a,b & c.

$a := 1 \qquad b := 0.5 \qquad c := 0.25$

$f(a,b,c,t) := a\cdot(t)^2 + b\cdot\exp(t) - c\cdot\tan(t)$



Now we'll define the data to fit; to more clearly show that Hooke-Jeeves function is working, we'll add a small constant to the y data before fitting it - this should show up in the plot as an offset curve.

Define the number of points

$N := 21 \qquad i := 0 .. N - 1$

Define the independent variable points

$t := \text{linspace}(-1, 1, N)$

$t^T = (-1 \quad -0.9 \quad -0.8 \quad -0.7 \quad -0.6 \quad -0.5 \quad -0.4 \quad -0.3 \quad -0.2 \quad -0.1 \quad 0 \quad 0.1 \quad 0.2 \quad 0.3 \quad 0.4 \quad 0.5 \quad 0.6 \quad 0.7 \quad 0.8 \quad 0.9 \quad 1)$

Define the dependent variable points

$y := \text{vec}\left(\overrightarrow{f(a,b,c,t)}\right) + 0.125$

$y^T = (1.698 \quad 1.453 \quad 1.247 \quad 1.074 \quad 0.93 \quad 0.815 \quad 0.726 \quad 0.663 \quad 0.625 \quad 0.613 \quad 0.625 \quad 0.663 \quad 0.725 \quad 0.813 \quad 0.925 \quad 1.0$

Define the fitting function

This takes a vector of a,b & c values as its input argument, making it suitable for passing the hookejeeves function.

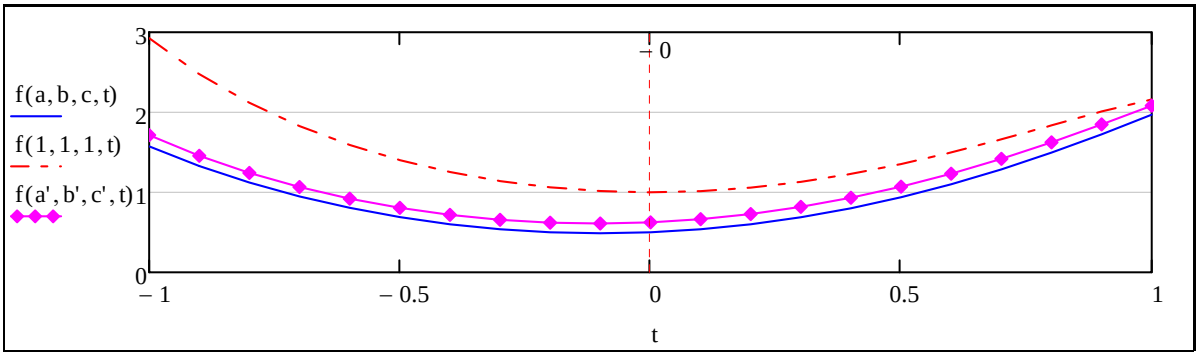
$$F(v) := \left| \begin{array}{l} (a \quad b \quad c) \leftarrow v^{T} \\ \left[\sum_{i=0}^{last(t)} \left(y_i - f\left(a, b, c, t_i\right) \right)^2 \right] \end{array} \right.$$

Find the original a, b and c parameters

$$res := hookejeeves(F, stack(1,1,1)) = \left[6.393 \quad 19 \quad \begin{pmatrix} 0.934 \\ 0.625 \\ 0.354 \end{pmatrix} \quad 1.112 \times 10^{-3} \right]$$

Plot the resultant curve

$$(a' \quad b' \quad c') := res_{0,2}{}^{T} \qquad (a' \quad b' \quad c') = (0.934 \quad 0.625 \quad 0.354)$$



$$y := vec\left(\overrightarrow{f(a', b', c', t)}\right)$$

$$y^{T} = (1.715 \quad 1.456 \quad 1.243 \quad 1.066 \quad 0.921 \quad 0.806 \quad 0.718 \quad 0.656 \quad 0.62 \quad 0.61 \quad 0.625 \quad 0.664 \quad 0.729 \quad 0.818 \quad 0.932 \quad 1.07$$