Chapter 17: Pattern Search for Unconstrained NLP

Sometimes it is inconvenient or not possible to know the first or second derivatives of the objective function in an unconstrained nonlinear programming problem. This can happen, for example, when the function value at some input point $\boldsymbol{x}$ is actually calculated by a simulation of a system, e.g. the simulation returns a measure of the flight characteristics of a new aircraft that is being designed.

In this case we can use heuristic *pattern search* methods which need only the ability to return the value of $f(\boldsymbol{x})$ for some input point $\boldsymbol{x}$. For this reason they are also known as *derivative-free*, *direct search*, or *black box* optimization methods. Of course pattern search methods can also be applied when the objective function is differentiable, but in that case we are ignoring the useful information in the derivatives and second derivatives. So pattern search methods are typically applied only when the derivatives are not available.

Quite a number of pattern search methods have been developed over the years. As an example of the genre, we will look at one of the original methods, known as *Hooke and Jeeves* after the original authors.

Hooke and Jeeves Pattern Search

This method was originally published in 1961. It involves two types of *moves*:

- *Exploratory search*. This is a very local search that looks for an improving direction in which to move. In some senses it is a crude search for the gradient direction.

- *Pattern move*. This is a larger search in the improving direction. Larger and larger moves are made as long as the improvement continues.

We'll look at each of these two types of moves separately before assembling them into the complete algorithm.

Exploratory Search

The main idea is to find *some* improving direction (not necessarily *the best* improving direction, which would be the gradient or anti-gradient direction). This is done by perturbing the current point by small amounts in each of the variable directions and observing whether the objective function value improves or worsens.

First we define the sizes of the perturbation steps that we will take in each dimension by setting up the perturbation vector $\boldsymbol{P}_0 = (\Delta x_1, \Delta x_2, \Delta x_3, \ldots \Delta x_n)$. Note that the perturbation step sizes do not all have to be equal, but in general they are all relatively small. Given some current point $\boldsymbol{x}^{(0)}$ and its associated objective function value $f(\boldsymbol{x}^{(0)})$ we can now perform an exploratory search around it as follows:

1. $\boldsymbol{x}^{(1)} \leftarrow \boldsymbol{x}^{(0)}$, i.e. copy $\boldsymbol{x}^{(0)}$ into $\boldsymbol{x}^{(1)}$. Initialize $f_{\text{best}} \leftarrow f(\boldsymbol{x}^{(0)})$.
2. For each variable x_j in turn:
    a. $\boldsymbol{x}^{(1)} \leftarrow \left(x_1^{(1)}, x_2^{(1)}, x_3^{(1)}, \ldots, x_j^{(1)} + \Delta x_j, \ldots, x_n^{(1)}\right)$, i.e. bump the jth variable <u>*up*</u> by its perturbation value.
    b. If $f(\boldsymbol{x}^{(1)})$ improves over current value of f_{best}: retain the perturbation, update $f_{\text{best}} \leftarrow f(\boldsymbol{x}^{(1)})$ and go to Step 2.a for the next variable.
    c. [$f(\boldsymbol{x}^{(1)})$ worsens or stays the same for the upwards perturbation]: $\boldsymbol{x}^{(1)} \leftarrow \left(x_1^{(1)}, x_2^{(1)}, x_3^{(1)}, \ldots, x_j^{(1)} - \Delta x_j, \ldots, x_n^{(1)}\right)$, , i.e. bump the variable <u>*down*</u> by its perturbation value.
    d. If $f\left(\boldsymbol{x}^{(1)}\right)$ improves over current value of f_{best}: retain the perturbation and update $f_{\text{best}} \leftarrow f(\boldsymbol{x}^{(1)})$. Otherwise discard the perturbation: $x_j^{(1)} \leftarrow x_j^{(0)}$.
    e. Go to Step 2.a for the next variable.
3. The improving direction is given by the vector $\boldsymbol{x}^{(1)} - \boldsymbol{x}^{(0)}$. If $\boldsymbol{x}^{(1)} = \boldsymbol{x}^{(0)}$ then the exploratory search has failed.

There are a few things to notice about the exploratory search algorithm:

- If the upward perturbation for a variable is successful, then the downward perturbation for that variable is not even attempted.
- The downward perturbation for a variable is tried only if the upward perturbation for that variable fails.
- It's possible that both the upward and the downward perturbations fail for a particular variable, in which case its value is not changed.
- The method does not try all possible combinations of upward and downward perturbations of the variables (there would be 2^n such combinations: too many to try). It is simply one pass through the list of variables. In the worst case where the upward perturbation fails for every variable, then it will try $2n$ combinations of perturbations, but it normally tries fewer than that. In the best case where every upward perturbation succeeds, it will try just n perturbations.

If *any* of the attempted perturbations succeeds in improving the value of the objective function, then the exploratory search has succeeded and the improving direction is given by the vector between the initial point $\boldsymbol{x}^{(0)}$ and the final point $\boldsymbol{x}^{(1)}$ output by the exploratory search. Now we can use this improving direction in the pattern move. If all of the perturbations fail to find an improved value of the objective function, then the exploratory search has failed: we'll see later what to do in that case.

Figure 17.1 gives some examples of exploratory searches.

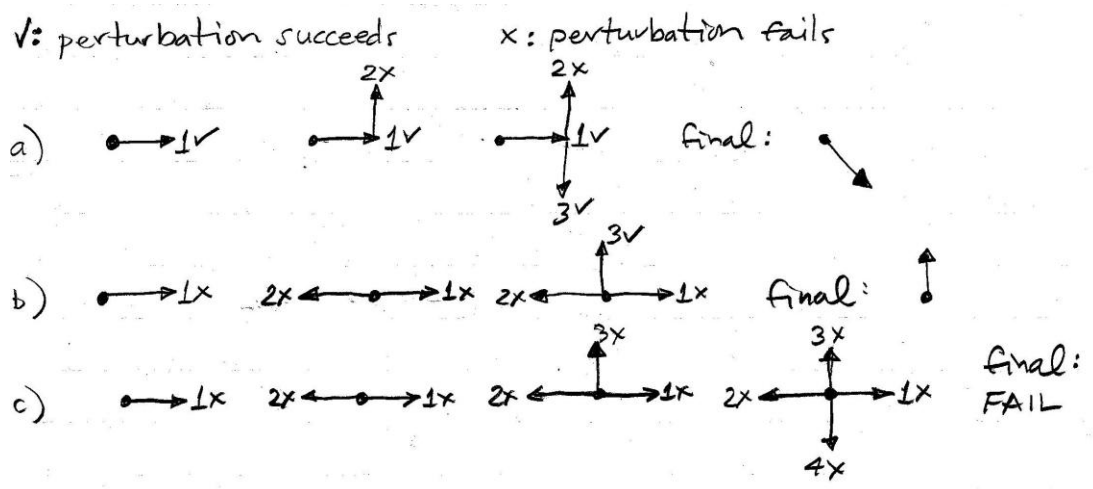

Figure 17.1: Examples of exploratory searches.

Pattern Move

All that the pattern move requires is two points: the current point $\boldsymbol{x}^{(0)}$ and some other point $\boldsymbol{x}^{(1)}$ that has a better value of the objective function. This gives the pattern move an improving direction to move in. A new point $\boldsymbol{x}^{(2)}$ is generated by moving from $\boldsymbol{x}^{(0)}$ through $\boldsymbol{x}^{(1)}$ as follows:

$$\boldsymbol{x}^{(2)} = \boldsymbol{x}^{(0)} + a\left[\boldsymbol{x}^{(1)} - \boldsymbol{x}^{(0)}\right]$$

where a is a positive *acceleration factor* that just multiplies the length of the improving direction vector given by $\boldsymbol{x}^{(1)} - \boldsymbol{x}^{(0)}$. A common choice is a=2, in which case the equation reduces to:

$$\boldsymbol{x}^{(2)} = 2\boldsymbol{x}^{(1)} - \boldsymbol{x}^{(0)}$$

Complete Algorithm

The complete algorithm requires 4 inputs in addition to the objective function to be optimized:

- A starting point $\boldsymbol{x}^{(0)}$,
- The value of the acceleration factor a,
- The initial perturbation vector $\boldsymbol{P}_0$,
- The perturbation tolerance vector $\boldsymbol{T} = (t_1, t_2, t_3, \ldots, t_n)$. As we will see later in the complete algorithm, this gives the smallest possible perturbation that can be considered for each variable, and is used to halt the algorithm.

The complete algorithm has 3 main parts: initialization, the start/restart routine, and the pattern move routine.

Initialization:

- choose the values for the starting point $\boldsymbol{x}^{(0)}$, acceleration factor a, perturbation vector $\boldsymbol{P}_0$, and perturbation tolerance vector $\boldsymbol{T}$. Initialize the current perturbation vector: $\boldsymbol{P} \leftarrow \boldsymbol{P}_0$.

Start/Restart Routine:

- Use an exploratory search around $\boldsymbol{x}^{(0)}$ to find an improved point $\boldsymbol{x}^{(1)}$ that has a better value of the objective function.
- IF the exploratory search fails (i.e. $\boldsymbol{x}^{(1)}$ does not give a better value of the objective function than $\boldsymbol{x}^{(0)}$) then:
    - Reset all of the perturbations to ½ their current size, i.e. $\boldsymbol{P} \leftarrow \boldsymbol{P}/2$.
    - If any member of $\boldsymbol{P}$ is now smaller than its corresponding perturbation tolerance in $\boldsymbol{T}$, then exit with $\boldsymbol{x}^{(0)}$ as the solution. Else go to *Start/Restart*.
- ELSE [$\boldsymbol{x}^{(1)}$ gives a better value of the objective function than $\boldsymbol{x}^{(0)}$, so we have an improving direction]:
    - Reset the perturbation vector to its original values: $\boldsymbol{P} \leftarrow \boldsymbol{P}_0$.
    - Go to *Pattern Move*.

Pattern Move:

- Obtain tentative $\boldsymbol{x}^{(2)}$ by a pattern move from $\boldsymbol{x}^{(0)}$ through $\boldsymbol{x}^{(1)}$.
- Obtain final $\boldsymbol{x}^{(2)}$ by an exploratory search around tentative $\boldsymbol{x}^{(2)}$.
- IF $f(\boldsymbol{x}^{(2)})$ is worse than $f(\boldsymbol{x}^{(1)})$ then:
    - Update points: $\boldsymbol{x}^{(0)} \leftarrow \boldsymbol{x}^{(1)}$. [$\boldsymbol{x}^{(1)}$ is the best point seen so far]
    - Go to *Start/Restart*.
- ELSE [$f(\boldsymbol{x}^{(2)})$ is better than or equal to $f(\boldsymbol{x}^{(1)})$]:
    - Update points: $\boldsymbol{x}^{(0)} \leftarrow \boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(1)} \leftarrow \boldsymbol{x}^{(2)}$.
    - Go to *Pattern Move*.

Note that pattern moves are repeated as long as they are successful, and usually become longer and longer. But as soon as a pattern move fails by producing an $f(\boldsymbol{x}^{(2)})$ that is worse than the previous best value of the objective function $f(\boldsymbol{x}^{(1)})$, then the pattern move is retracted and we go back to an exploratory search around the best point seen so far.

Note also that a pattern move first obtains a tentative $\boldsymbol{x}^{(2)}$ and then finalizes $\boldsymbol{x}^{(2)}$ by an exploratory search. This helps the search to "curve", as we will see in the upcoming example. Lastly note how the algorithm stops: an exploratory search around the best point seen so far fails at all sizes of perturbation, even the smallest as specified by the perturbation tolerances in $\boldsymbol{T}$. This means that the algorithm cannot find an improving direction away from the best point, hence it must be the optimum point. This test can be fooled though, as we will see later.

Example:

Minimize $f(\boldsymbol{x}) = 3x_1^2 + x_2^2 - 12x_1 - 8x_2$

Initialize:

- Initial point: $\boldsymbol{x}^{(0)} = (1, 1)$. $f(\boldsymbol{x}^{(0)}) = -16$.
- Acceleration factor $a = 2$.
- Perturbation vector $\boldsymbol{P}_0 = (0.5, 0.5)$.
- Perturbation tolerance vector $\boldsymbol{T} = (0.1, 0.1)$.
- $\boldsymbol{P} \leftarrow \boldsymbol{P}_0$.

  Note that these are not very good choices for $\boldsymbol{P}_0$ and $\boldsymbol{T}$. They are chosen in this case just so that the algorithm terminates after a small number of steps. The elements in $\boldsymbol{T}$ would normally be much smaller.

Start/Restart:

- $f_{\text{best}} = f(\boldsymbol{x}^{(0)}) = -16$.
- Try $\boldsymbol{x}^{(1)} = (1.5, 1)$. $f(\boldsymbol{x}^{(1)}) = -18.25$, so keep the perturbation and update $f_{\text{best}} = -18.25$.
- Try $\boldsymbol{x}^{(1)} = (1.5, 1.5)$. $f(\boldsymbol{x}^{(1)}) = -21$, so keep the perturbation and update $f_{\text{best}} = -21$.

  The steps in the exploratory search are shown in this first Start/Restart, but are omitted from here forward.

Pattern Move from $\boldsymbol{x}^{(0)} = (1, 1)$ through $\boldsymbol{x}^{(1)} = (1.5, 1.5)$:

- Tentative $\boldsymbol{x}^{(2)} = 2(1.5, 1.5) - (1, 1) = (2, 2)$. $f(2, 2) = -24$.
- Final $\boldsymbol{x}^{(2)}$ after exploratory search around tentative $\boldsymbol{x}^{(2)}$ is (2.0, 2.5). $f(\boldsymbol{x}^{(2)}) = -25.75$ is better than $f(\boldsymbol{x}^{(1)}) = -21$ so the move is accepted.
- Update points: $\boldsymbol{x}^{(0)} \leftarrow \boldsymbol{x}^{(1)} = (1.5, 1.5)$ and $\boldsymbol{x}^{(1)} \leftarrow \boldsymbol{x}^{(2)} = (2.0, 2.5)$.

Pattern Move from $\boldsymbol{x}^{(0)} = (1.5, 1.5)$ through $\boldsymbol{x}^{(1)} = (2.0, 2.5)$:

- Tentative $\boldsymbol{x}^{(2)} = 2(2.0, 2.5) - (1.5, 1.5) = (2.5, 3.5)$. $f(2.5, 3.5) = -27$.
- Final $\boldsymbol{x}^{(2)}$ after exploratory search around tentative $\boldsymbol{x}^{(2)}$ is (2.0, 4.0). $f(\boldsymbol{x}^{(2)}) = -28$ is better than $f(\boldsymbol{x}^{(1)}) = -25{,}75$ so the move is accepted.
- Update points: $\boldsymbol{x}^{(0)} \leftarrow \boldsymbol{x}^{(1)} = (2.0, 2.5)$ and $\boldsymbol{x}^{(1)} \leftarrow \boldsymbol{x}^{(2)} = (2.0, 4.0)$.

Pattern Move from $\boldsymbol{x}^{(0)} = (2.0, 2.5)$ through $\boldsymbol{x}^{(1)} = (2.0, 4.0)$:

- Tentative $\boldsymbol{x}^{(2)} = 2(2.0, 4.0) - (2.0, 2.5) = (2.0, 5.5)$. $f(2.0, 5.5) = -25.75$.
- Final $\boldsymbol{x}^{(2)}$ after exploratory search around tentative $\boldsymbol{x}^{(2)}$ is (2.0, 5.0). $f(\boldsymbol{x}^{(2)}) = -27$ is worse than $f(\boldsymbol{x}^{(1)}) = -28$ so the move is rejected.
- Update points: $\boldsymbol{x}^{(0)} \leftarrow \boldsymbol{x}^{(1)} = (2.0, 4.0)$.

Start/Restart:

- Exploratory search around $\boldsymbol{x}^{(0)}$ = (2.0, 4.0) fails at all levels of perturbation size.
- Exit with solution $\boldsymbol{x}^{(0)}$ = (2.0, 4.0) and f($\boldsymbol{x}^{(0)}$) = -28.

The table below shows the sequence of points in the solution to our sample problem, and these are plotted in Figure 17.2.

a: $f(1, 1) = -16$	e: $f(2.5, 3.5) = -27$
b: $f(1.5, 1.5) = -21$	f: $f(2, 4) = -28$ [*eventual solution*]
c: $f(2, 2) = -24$	g: $f(2, 5.5) = -25.75$
d: $f(2, 2.5) = -25.75$	h: $f(2,5) = -27$

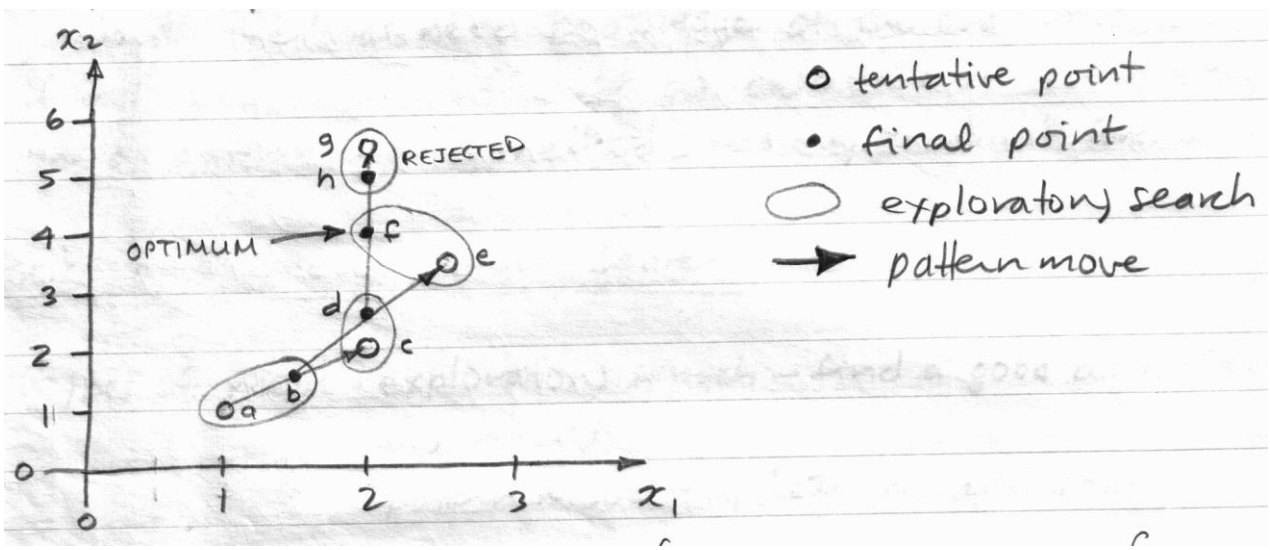

Figure 17.2: Plot of the example.

Note how the exploratory search around each tentative $\boldsymbol{x}^{(2)}$ allows the search to "curve" around corners.

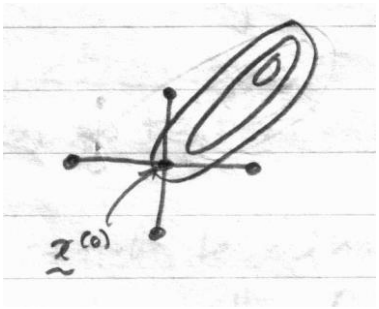

Figure 3: Failure of the exploratory search.

As mentioned earlier, the stopping conditions in this heuristic method can be fooled. This happens when the exploratory search "steps over" a feature, and is therefore unable to find an improving direction even when one exists. This can happen as shown in Figure 17.3, in which the contour lines taper to an optimum point in the small circle at the upper right. However, all of the individual exploratory search steps around $\boldsymbol{x}^{(0)}$ fail, even though an improving direction is available. The simple Hooke and Jeeves exploratory search is unable to find that improving direction, but remember that it is a heuristic after all. However it is easy to improve and extend the basic method. For example, if the exploratory search fails, it could be augmented by testing a few random search directions from $\boldsymbol{x}^{(0)}$.

Other Pattern Search Methods

There are a variety of other pattern search methods. One of the best known is the *Nelder-Mead method*, sometimes (confusingly) called the *downhill simplex method*. This method maintains a polytope of n+1 points in an n-dimensional problem. In two dimensions this polytope is a triangle. The objective function value is evaluated at each of the points, and hence some improving directions can be identified. Movements are then made by "flipping" the triangle to move the worst point, expanding or shrinking the triangle etc. An excellent animation of the Nelder-Mead algorithm can be seen on Wikipedia at http://en.wikipedia.org/wiki/File:Nelder_Mead2.gif.

The Matlab Global Optimization Toolbox includes (as of 2014) 3 pattern search methods: generalized pattern search, generating set search, and mesh adaptive search. The optimization package for the free Matlab-like Octave system includes a Nelder-Mead implementation.

An interesting sub-field of pattern search research is determining how to find a good solution for an unconstrained problem with the smallest number of function evaluations. This is important when each function evaluation is expensive, either in time or actual money. It could be that finding $f(\boldsymbol{x})$ for some value of $\boldsymbol{x}$ requires a long-running simulation (expensive in terms of time), or it requires the construction and testing of a physical prototype (expensive in terms of both time and money). Some of work along these lines can be found under the search term "efficient global optimization".