Pattern Search Methods
for
Engineering Optimization

Virginia Torczon
Department of Computer Science
College of William & Mary
Williamsburg, VA 23187–8795

va@cs.wm.edu

Collaborators:

J. E. Dennis, Jr., Rice University
Robert Michael Lewis, ICASE, NASA Langley
David Serafini, Rice University
Michael Trosset, University of Arizona

The general optimization problem:

$$\min_{x \in \Omega \subseteq \mathbf{R}^n} \quad f(x)$$

Assume (for simplicity) that $\Omega \equiv \mathbf{R}^n$; pattern search methods also exist for bound constraints

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & \ell \le x \le u \end{array}$$

and for general linear inequality constraints

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & \ell \le Ax \le u. \end{array}$$

Features of pattern search methods:

⋆ robust behavior in practice

⋆ observed behavior backed by convergence analysis

- do not require derivatives or approximations to derivatives
- do not require numeric function values
- simple to implement
- easy to use
- strong ties to experimental design and response surface methodology (RSM)
- heuristics motivated by desire to capture direction of “steep” descent
- straightforward to customize

Why so popular in the user community?

Applicable when:

- f is nondifferentiable
- derivatives are either not available or not reliable
- evaluation of f is inaccurate
- f is highly nonlinear
- only subjective (rank or order) information is available
- want to explore the design space (concerned about getting trapped at local minimizers)

Outline:

- generalized pattern search
- an example
- highlights of analysis
- computational characteristics
- pattern search versus genetic algorithms
- summary

Generalized pattern search:

Given $x_0 \in \mathbf{R}^n$, $f(x_0)$, $P_0 \in \mathbf{R}^{n \times p}$, and $\Delta_0 > 0$, for $k = 0, 1, \ldots$ until *convergence* do {

1. Find a step s_k using `Exploratory Moves`(Δ_k, P_k).

2. If $f(x_k + s_k)$ is better than $f(x_k)$, then

$$x_{k+1} = x_k + s_k.$$

   Otherwise,

$$x_{k+1} = x_k.$$

3. `Update`(Δ_k, P_k)

}

Key features of the `Exploratory Moves`:

- *Any* trial step s_k^i *must* satisfy: $s_k^i \in \Delta_k P_k$ (i.e., the trial steps must be defined by the pattern)

- *Any* step $s_k^i \in \Delta_k P_k$ for which $f(x_k + s_k^i)$ is better than $f(x_k)$ is acceptable.

- *Must* have a "fall back" strategy with a minimum of $n+1$ points.

- Keep x_k—and reduce Δ_k—*only* if no step s_k^i defined by the fall back strategy satisfies the condition that $f(x_k + s_k^i)$ is better than $f(x_k)$.

Note:

The pattern P_k must satisfy certain mild algebraic and geometric conditions, as must the update of Δ_k.

Algorithmic features:

- admit *wide* range of heuristics—both old and new—and yet preserve robustness
  (1997 Torczon/Trosset, submitted to Technometrics)
  (1997 Dennis/Serafini/Torczon, in preparation)
  (i.e., the approximation model management strategy of Dennis/Torczon)

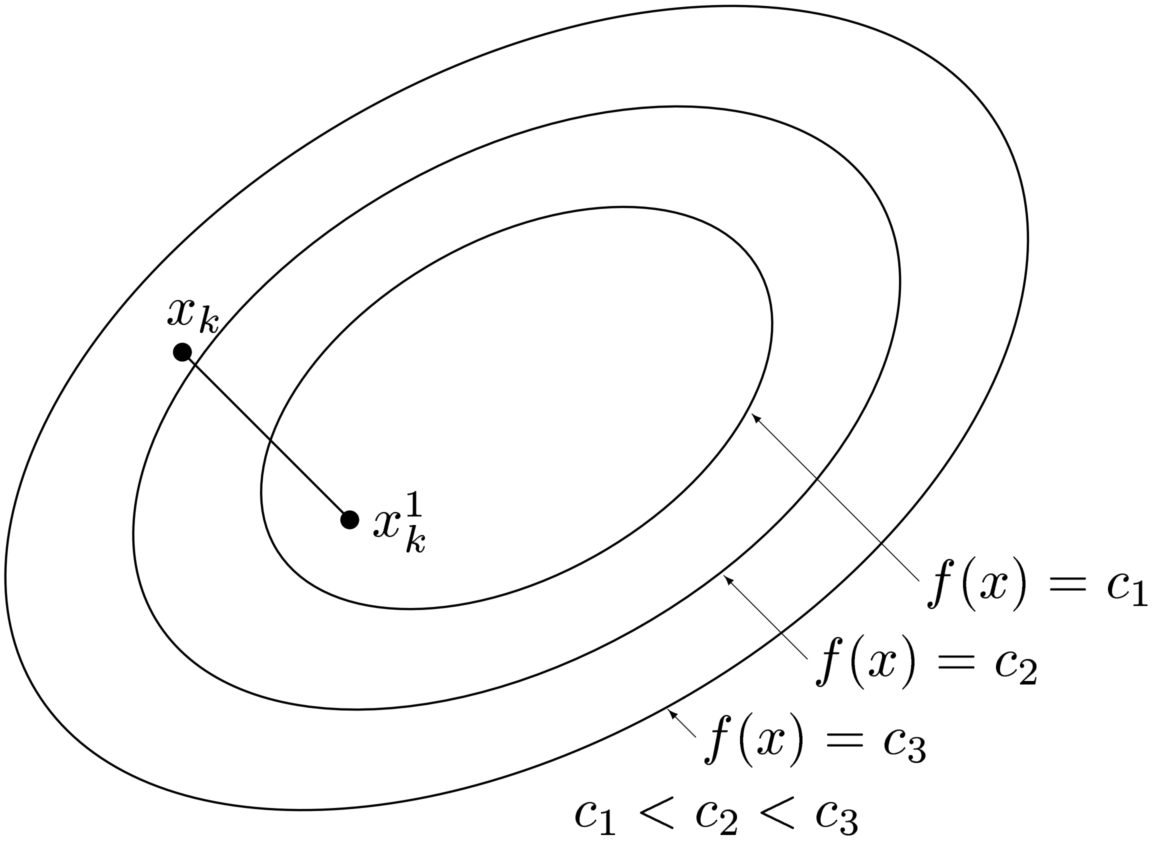
- can be quite frugal:

  – best case, 1 evaluation of f/iteration;
  – worst case (the fall back strategy), $n + 1$ evaluations of f/iteration

  (1996 Lewis/Torczon, submitted to Mathematical Programming)

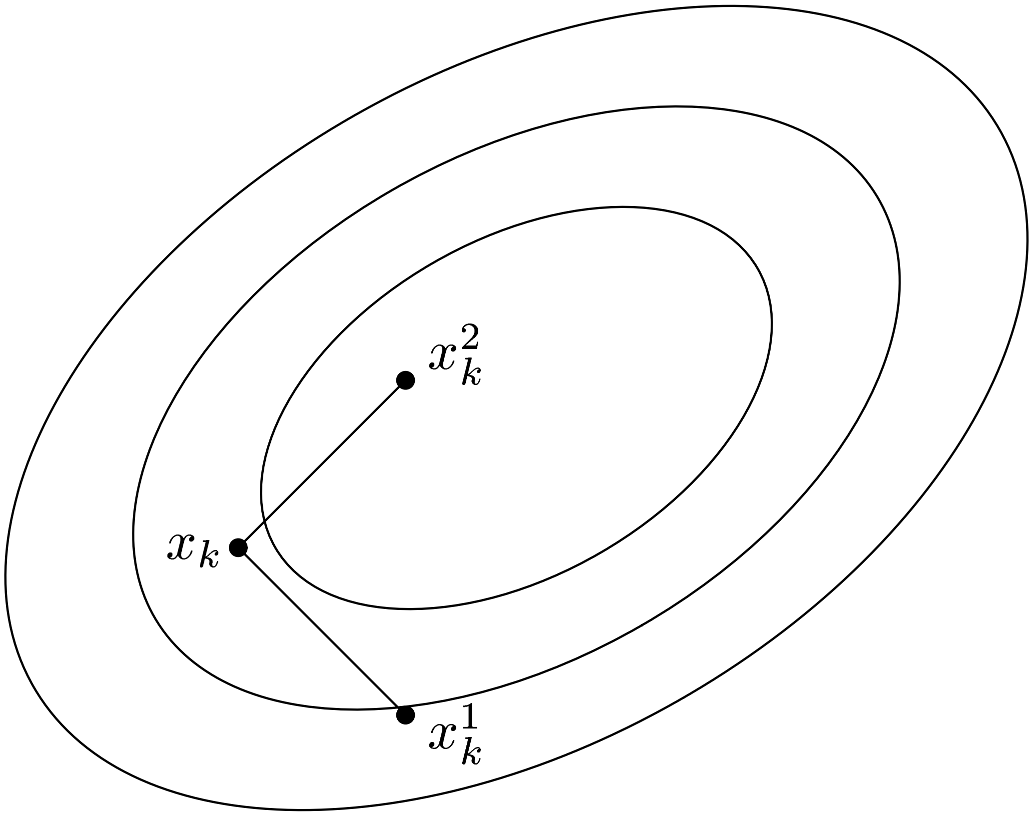
- embarrassingly parallel
  (1991 Dennis/Torczon, SIAM J. on Optimization)

Scenario 1:



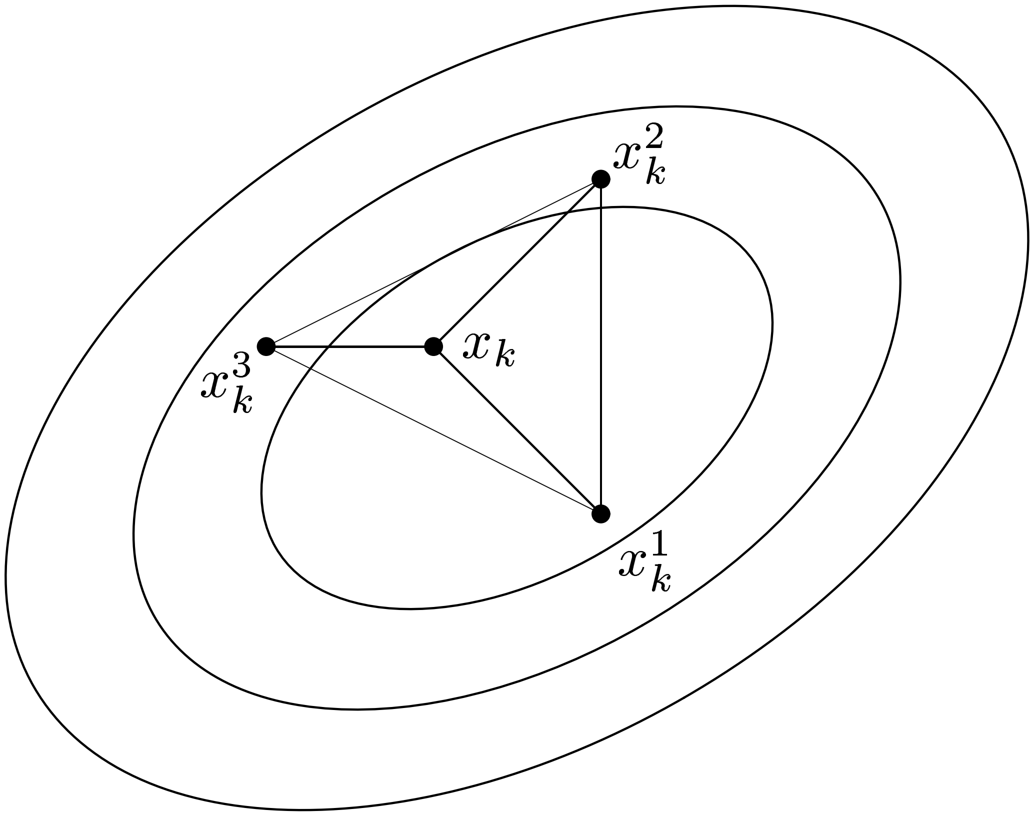
Try $x_k^1 = x_k + s_k^1$.
Accept x_k^1.

Scenario 2:



Try x_k^1, then $x_k^2 = x_k + s_k^2$.
Accept x_k^2.

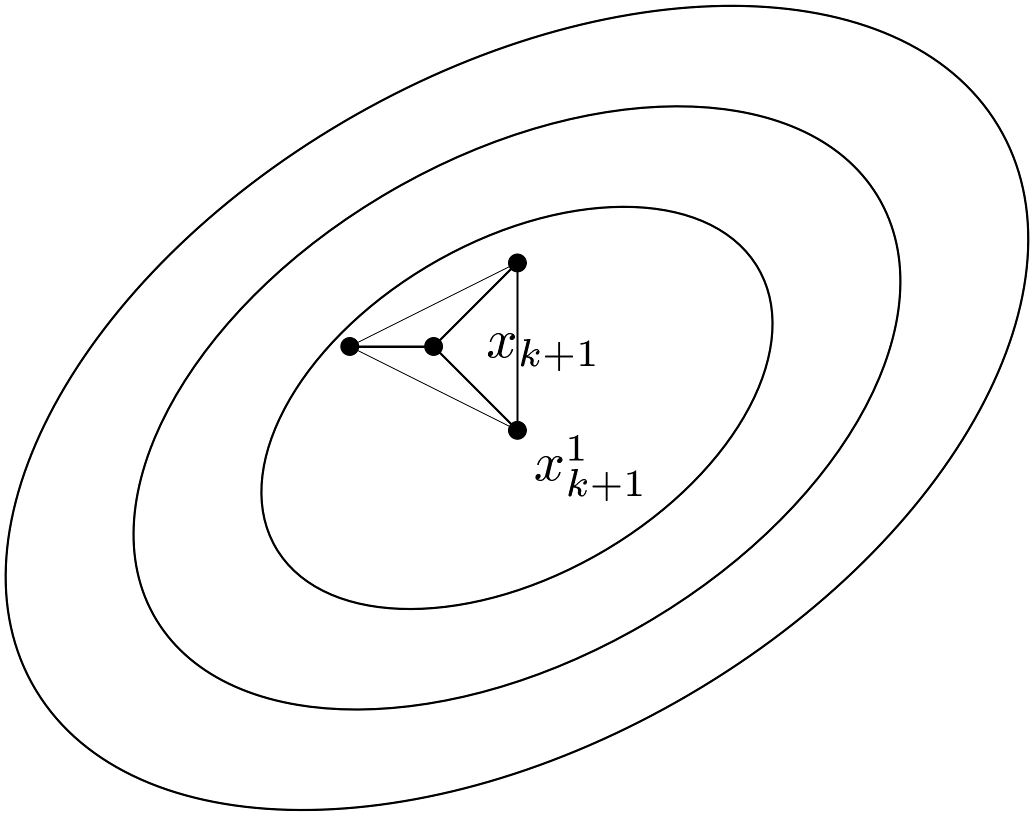
Scenario 3:



Try x_k^1, then x_k^2, and finally $x_k^3 = x_k + s_k^3$.
Reject all 3 steps and reduce Δ_k.

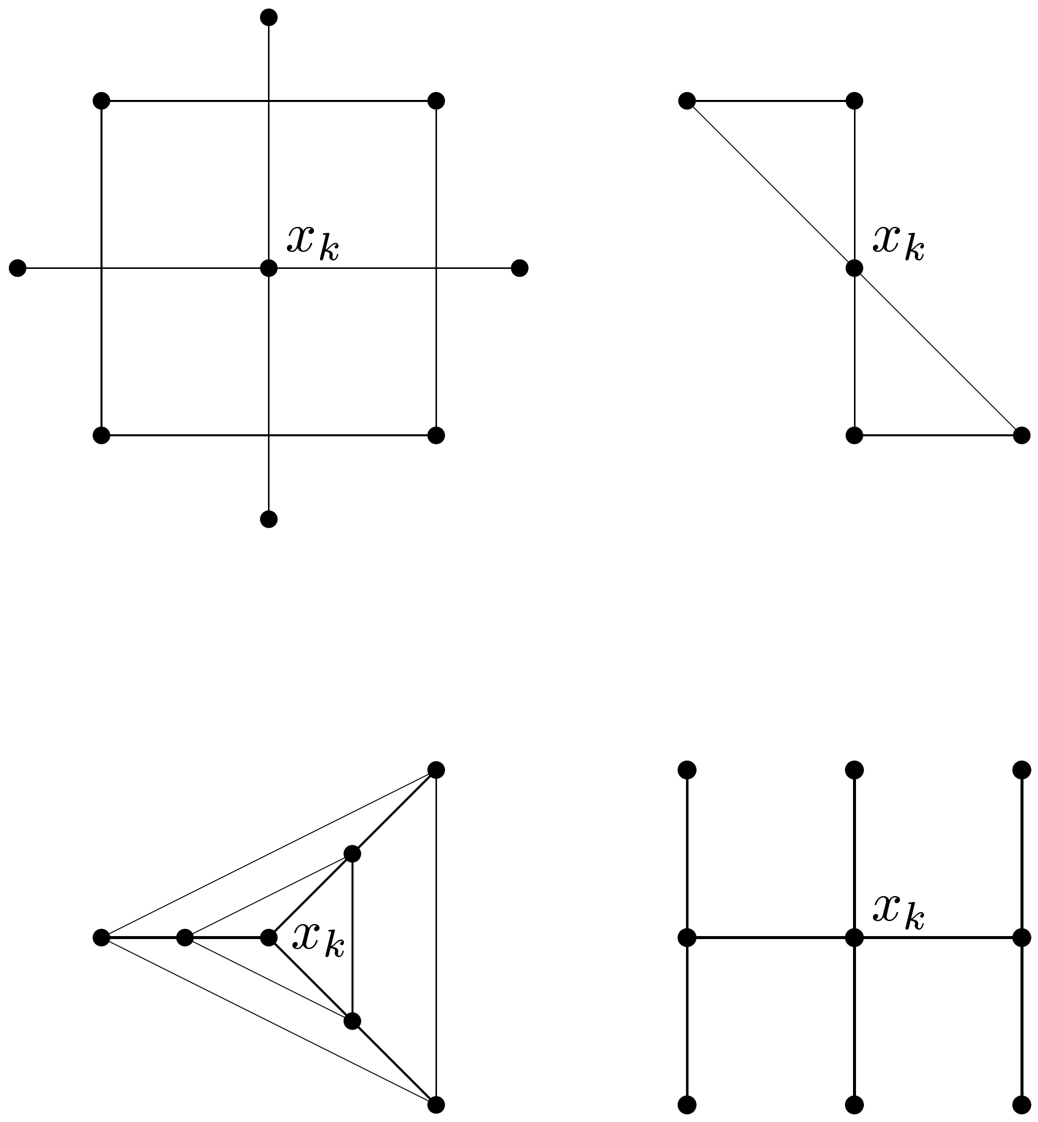
$$P_k = \begin{bmatrix} 2 & 2 & -2 & 0 \\ -2 & 2 & 0 & 0 \end{bmatrix}$$

Scenario 3, continued:

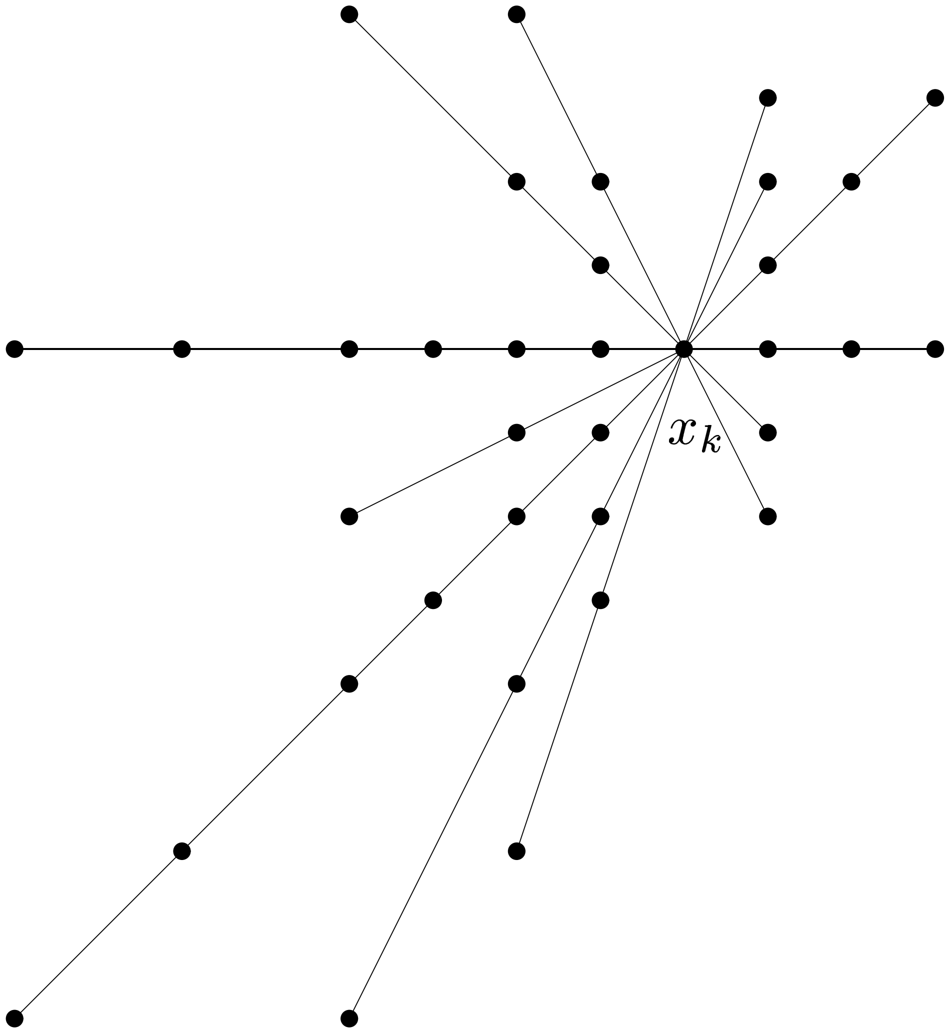


Repeat the search with $x_{k+1} = x_k$,
$P_{k+1} = P_k$, and $\Delta_{k+1} = \frac{1}{2}\Delta_k$:
Try and accept x_{k+1}^1.

Some classical patterns:



A more aggressive pattern:



$$P_k \quad = \quad \left[\begin{array}{cccccccccc} 2 & 2 & -2 & -2 & 1 & 1 & \ldots & -8 & -4 & 0 \\ 0 & 2 & 0 & -2 & 0 & 1 & & 0 & 4 & 0 \end{array} \right]$$

An important distinction:

Global convergence versus convergence to a global minimizer:

- **Global convergence:** Convergence to a stationary point or local minimizer from an arbitrary starting point.

- **Convergence to a global minimizer:** Convergence to a global minimizer over the feasible region.

In linear and convex programming, one gets global convergence to a global minimizer. In nonconvex nonlinear optimization, this is not necessarily the case.

There are ways to insure global convergence and it is relatively easy to achieve in practice. Convergence to a global minimizer is both difficult to insure and computationally expensive.

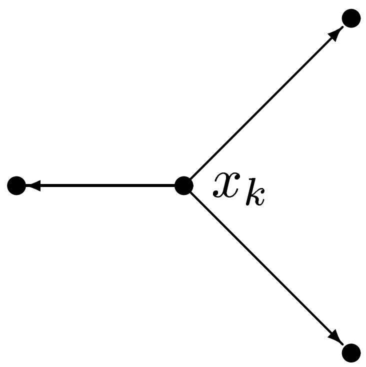
Classic outline for global convergence analysis:

1. descent method

2. gradient-related method

3. step length control

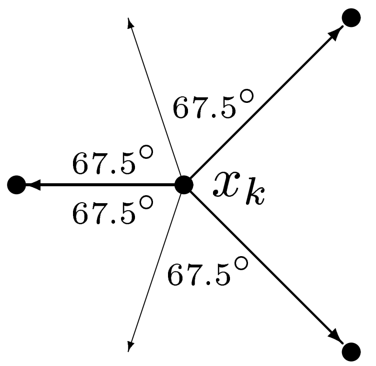
At least one direction of descent:



$\Longrightarrow$ pattern search methods are descent methods.

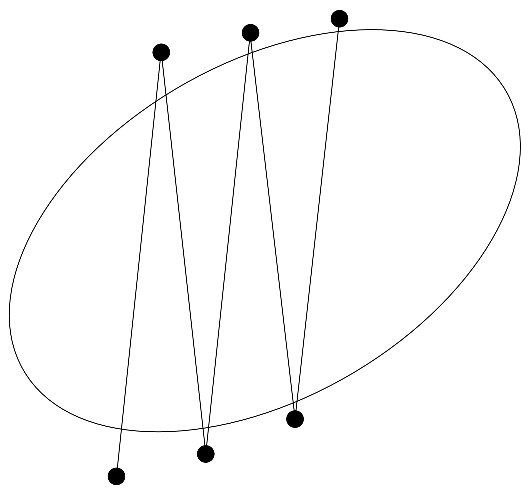
A lower bound on distance from $-\nabla f(x_k)$:

(In this example, for instance, the three search directions can be at most 67.5° from $-\nabla f(x_k)$.)



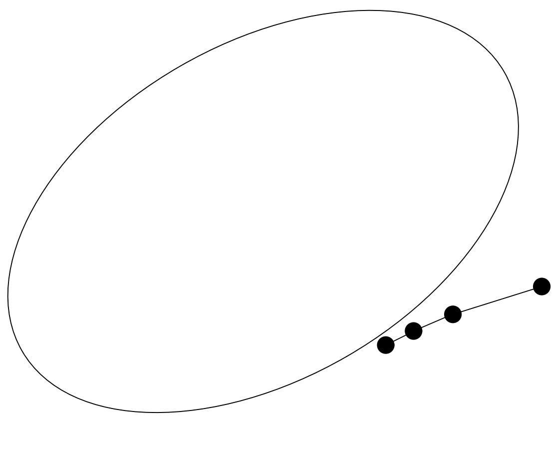
$\Longrightarrow$ pattern search methods are gradient-related methods.

Possible pathologies:



decrease too small
relative to the
length of the step

$\Longrightarrow$ need for step length control



decrease too small
relative to the
norm of the gradient

If one has ∇f then the analysis is straightforward:

1. choose direction *related* to $-\nabla f(x_k)$

2. force step to satisfy *sufficient* amount of decrease predicted by $\|\nabla f(x_k)\|$ and $\|s_k\|$

Pattern search methods:

$-$ do not have ∇f or an approximation thereof

$-$ accept $x_{k+1} = x_k + s_k$ if x_{k+1} is better than x_k (i.e., *simple*, as opposed to *sufficient*, decrease)

$+$ include the classical experimental designs (patterns)

$+$ if

- the pattern contains at least $n+1$ points and
- the pattern is "safeguarded"

then enough structure to prove global convergence

Bottom line:

Pattern search methods are *robust*; i.e.,

$$\lim_{k \to +\infty} \|\nabla f(x_k)\| \quad = \quad 0$$

Recent results:

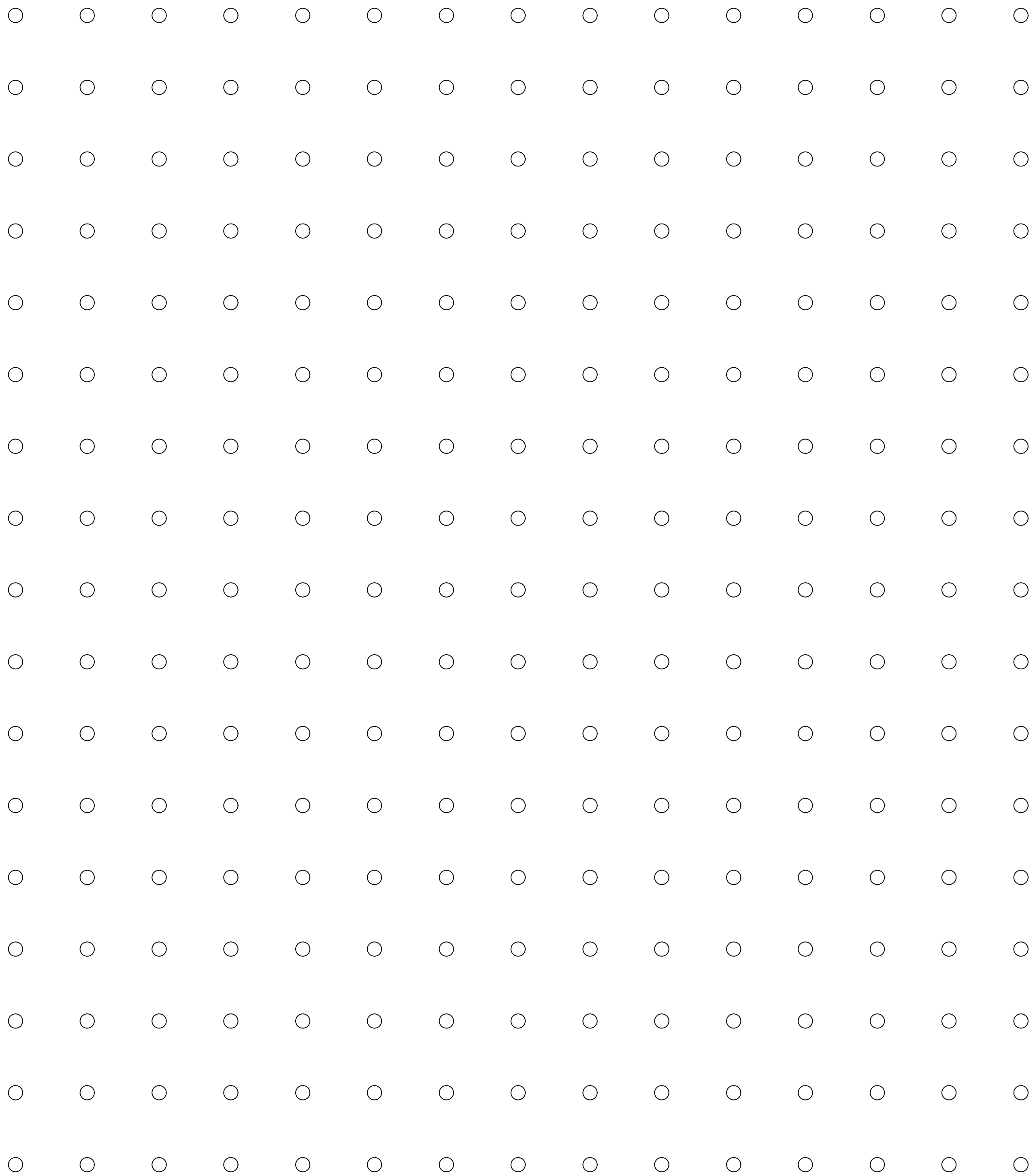
1997 Torczon
On the convergence of pattern search algorithms
SIAM Journal on Optimization

To appear McKinnon
Convergence of the Nelder-Mead simplex method to a non-stationary point
SIAM Journal on Optimization

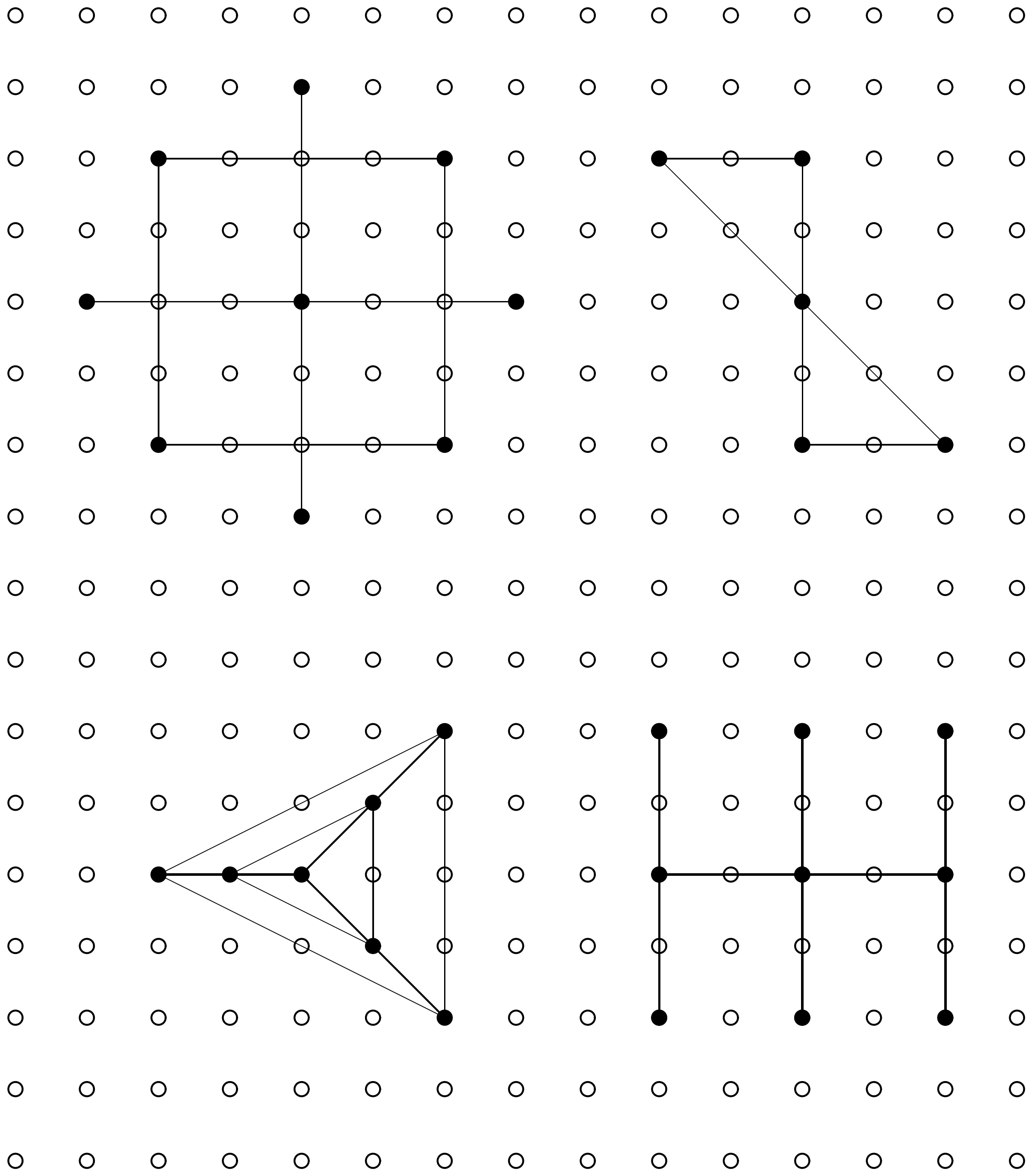
To appear Lewis/Torczon
Pattern search algorithms for bound constrained minimization
SIAM Journal on Optimization

In preparation Lewis/Torczon
Pattern search algorithms for linearly constrained minimization

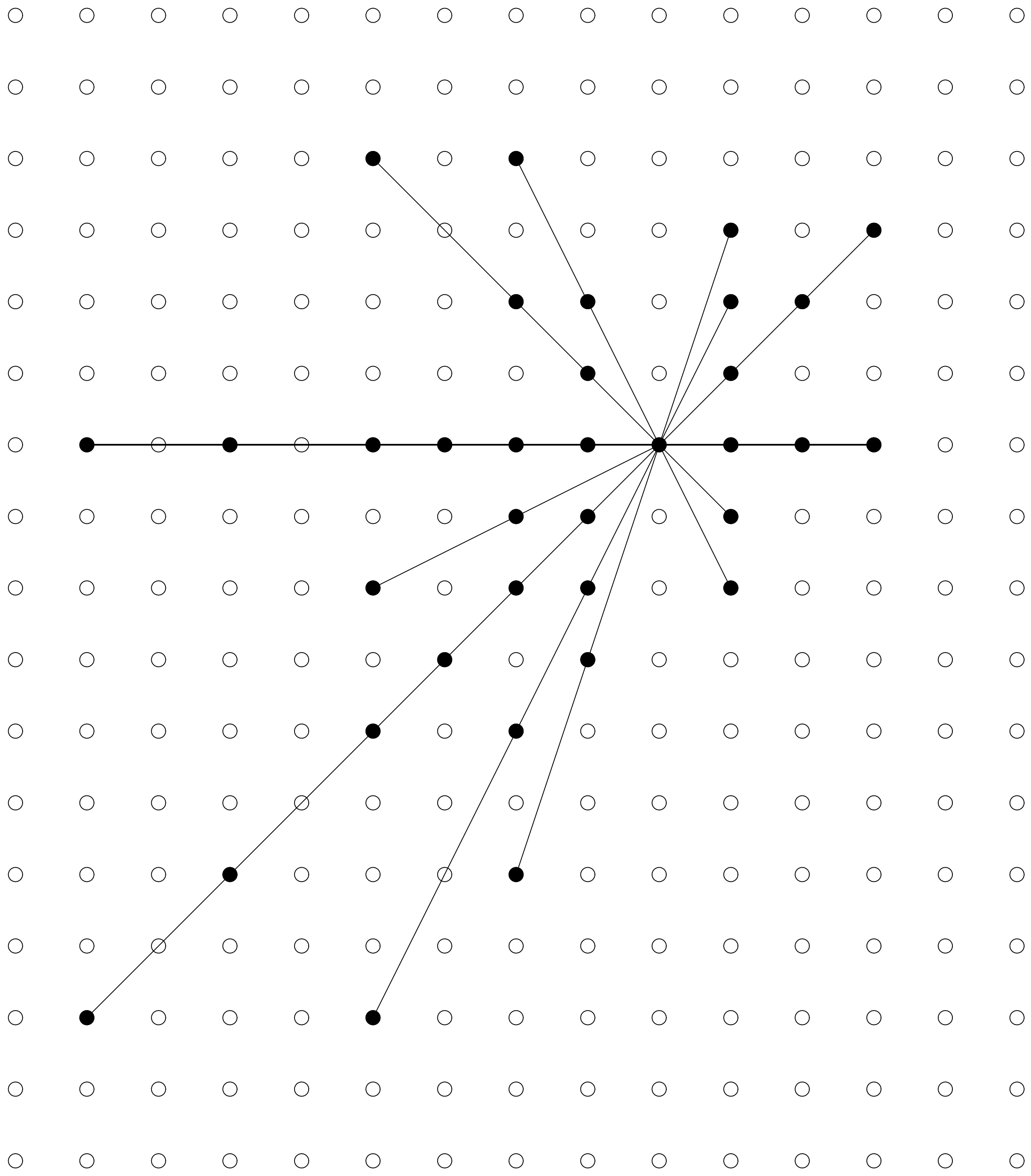
Key feature of the analysis: the lattice



Key feature of the analysis: the lattice



Key feature of the analysis: the lattice



The Nelder-Mead simplex method can fail
(for $n > 1$)

- anecdotal evidence
  "the simplex can collapse"

- experimental evidence
  Torczon, 1989

- analysis
  McKinnon, to appear

Bottom line:

The Nelder-Mead simplex method is *not* robust; i.e., may converge to a point x_* for which $\|\nabla f(x_*)\| \neq 0$.

The problem:

Cannot guarantee—even for convex, differentiable functions of two variables—that the search direction is gradient-related because the method does *not* prevent the simplex from becoming degenerate.

Pattern search methods

Advantages:

- Simple to implement.
- Simple to parallelize: distribute the pattern P_k.
- Can be used in the absence of numerical objective and constraints (e.g., when only rank or ordinal values are applicable).

⋆ Robustness/convergence analysis.

Purported advantages:

- Explores the design space (i.e., it can sample widely).

Pattern search methods

Disadvantages:

- Can be slow.
- Less efficient as the number of design variables grows.
- Optimal level of scaling not known *a priori*.

Question:

How do pattern search methods compare to genetic algorithms?

Genetic algorithms:

Given an initial collection of candidate solutions X_0 ("population"), for $k = 0, 1, \ldots$ until *halted* do $\{$

Find a new set X_{k+1} of potential solutions:

1. Draw from X_k according to a distribution weighted in favor of the "better" members ("reproduction").
2. Encode the elements of X_k as bit strings Y_k.
3. Randomly choose pairs from Y_k and swap randomly chosen substrings ("cross-over").
4. Randomly flip bits in randomly chosen elements of Y_k ("mutation").
5. Map the resulting collection of bit strings Y_{k+1} to X_{k+1}.

$\}$

One can think of this process as generating a sequence $\{x_k\}$ where x_k is the best element of X_k.

Genetic algorithms

Advantages:

- Simple to implement.
- Simple to parallelize: distribute the population X_k.
- Can be used in the absence of numerical objective and constraints (e.g., when only rank or ordinal values are applicable).
- Applicable to nonlinear problems with discrete variables.

Purported advantages:

- Explores the design space (i.e., it can sample widely).

Genetic algorithms

Disadvantages:

- Slow and inefficient.
- Can be ineffective as the number of design variables or constraints grows.
- No satisfactory convergence analysis.

Pattern search versus genetic algorithms

Implementations:

- `PGAPack`: Parallel Genetic Algorithm Package
  David Levine, Argonne National Laboratory

- `PDS`: Parallel Direct Search
  Virginia Torczon, Center for Research in Parallel Computation

Test functions:

- Three of the test functions from the DeJong test suite

Settings:

- default (for both packages)

- *no* tuning for either package

DeJong test function F1 (population/iteration = 100)

$$f_1(x) = \sum_{i=1}^{3} x_i^2, \qquad -5.12 \leq x_i \leq 5.11$$

PGAPack			PDS			optimal	
10 bits for the gray-coding			tol = 1.0D-03; x_0 = (5.11,5.11,5.11)				
best x	best f	# iter.	best x	best f	# iter.	x_*	f_*
1.00D-02	6.50D-03	100	-3.52D-04	1.61D-07	16	0	0
-8.00D-02			1.36D-04			0	
-4.00D-02			1.36D-04			0	

DeJong test function F1 (population/iteration = 100)

$$f_1(x) = \sum_{i=1}^{3} x_i^2, \qquad -5.12 \leq x_i \leq 5.11$$

PGAPack			PDS			optimal	
31 bits for the gray-coding			tol = 1.0D-08; x_0 = (5.11,5.11,5.11)				
best x	best f	# iter.	best x	best f	# iter.	x_*	f_*
0.30D+00	8.84D-02	100	5.96D-10	1.06D-18	31	0	0
-2.58D-03			5.96D-10			0	
1.89D-02			5.96D-10			0	

DeJong test function F2 (population/iteration = 100)

$$f_2(x) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2, \qquad -2.048 \le x_i \le 2.047$$

PGAPack			PDS			optimal	
12 bits for the gray-coding			tol = 1.0D-03; x_0 = (-1.2,1)				
best x	best f	# iter.	best x	best f	# iter.	x_*	f_*
0.89D+00 0.79D+00	1.14D-02	100	0.98D+00 0.96D+00	2.50D-04	74	1 1	0

PGAPack			PDS			optimal	
31 bits for the gray-coding			tol = 1.0D-08; x_0 = (-1.2,1)				
best x	best f	# iter.	best x	best f	# iter.	x_*	f_*
0.49D+00 0.24D+00	2.55D-01	100	0.99D+00 0.99D+00	5.90D-06	100	1 1	0

DeJong test function F3 (population/iteration = 100)

$$f_3(x) = \sum_{i=1}^{5} \text{integer}(x_i), \qquad -5.12 \leq x_i \leq 5.11$$

PGAPack			PDS			optimal	
10 bits for the gray-coding			tol = 1.0D-03; $x_0 = (5.11, \ldots, 5.11)$				
best x	best f	# iter.	best x	best f	# iter.	x_*	f_*
-5.03D+00	-2.70D+01	100	-5.01D+00	-3.00D+01	40	inf.	-30
-4.68D+00			-5.01D+00			many	
-4.04D+00			-5.01D+00				
-4.69D+00			-5.01D+00				
-5.03D+00			-5.01D+00				

DeJong test function F3 (population/iteration = 100)

$$f_3(x) = \sum_{i=1}^{5} \text{integer}(x_i), \qquad -5.12 \leq x_i \leq 5.11$$

PGAPack			PDS			optimal	
31 bits for the gray-coding			tol = 1.0D-08; x_0 = (5.11,. . . ,5.11)				
best x	best f	# iter.	best x	best f	# iter.	x_*	f_*
-4.32D+00	-2.80D+01	100	-5.01D+00	-3.00D+01	57	inf.	-30
-5.02D+00			-5.01D+00			many	
-5.09D+00			-5.01D+00				
-4.20D+00			-5.01D+00				
-5.02D+00			-5.01D+00				

Modified DeJong test function F1.12 (population/iteration = 100)

$$f_1(x) = \sum_{i=1}^{12} x_i^2, \qquad -5.12 \le x_i \le 5.11$$

PGAPack			PDS			optimal	
10 bits for the gray-coding			tol = 1.0D-03; x_0 = (5.11,. . . ,5.11)				
best x	best f	# iter.	best x	best f	# iter.	x_*	f_*
0.24D+00	3.71D+00	100	-1.50D-02	3.94D-02	100	0	0
-0.87D+00			-1.50D-02			0	
0.20D+00			6.25D-04			0	
-0.13D+00			6.25D-04			0	
-0.86D+00			9.43D-02			0	
-0.20D+00			6.25D-04			0	
0.27D+00			6.25D-04			0	
-0.24D+00			-1.50D-02			0	
6.00D-02			6.25D-04			0	
1.39D+00			9.43D-02			0	
-3.00D-02			9.43D-02			0	
-0.85D+00			0.11D+00			0	

Modified DeJong test function F1.12 (population/iteration = 400)

$$f_1(x) = \sum_{i=1}^{12} x_i^2, \qquad -5.12 \le x_i \le 5.11$$

PGAPack			PDS			optimal	
10 bits for the gray-coding			tol = 1.0D-03; x_0 = (5.11,. . . ,5.11)				
best x	best f	# iter.	best x	best f	# iter.	x_*	f_*
0.73D+00	9.31D-01	100	-6.10D-07	4.32D-10	100	0	0
-0.48D+00			-6.10D-07			0	
-5.00D-02			1.46D-05			0	
0.47D+00			-6.10D-07			0	
9.00D-02			-6.10D-07			0	
-0.24D+00			-6.10D-07			0	
0.28D+00			-6.10D-07			0	
-0.31D+00			-6.10D-07			0	
0.23D+00			1.46D-05			0	
-0.31D+00			-6.10D-07			0	
-9.00D-02			-6.10D-07			0	
-0.26D+00			-6.10D-07			0	

What to expect from pattern search methods:

- useful exploratory tool
  (wide applicability, ease of use)

- good global behavior
  (they are gradient-related methods)

- slow local convergence
  (no second-order information)

- potential for large number of function evaluations
  (they are sampling methods)

- effective for problems with a relatively small number of variables
  (curse of dimensionality)

When to use pattern search methods:

- evaluation of f is inaccurate
- derivatives of f are either not available or not reliable
- f is highly nonlinear
- f is nondifferentiable
- the first partial derivatives of f are discontinuous
- only rank (order) information is available
- want a "wider" search of the design space

Problems solved:

- chemical process control
- seismic inversion
- hearing recovery after exposure to loud noise
- control of robotic arm
- airfoil design
- vibration suppression
- helicopter rotor blade design
-
-
-

Coming soon:

Release 2.0 of PDS with extensions to handle bound constraints.

- written in Fortran 77
- can be run sequentially
- can be run in parallel on any platform that supports MPI

Send email to: va@cs.wm.edu.

Also look for postings on a new and improved web page:
http://www.cs.wm.edu/~va

PDS will soon be on NEOS, the Network-Enabled Optimization System from Argonne National Laboratory. Check out the web page:
http://www.mcs.anl.gov/home/otc/Server