AN EXTENDED CONTINUOUS NEWTON METHOD[†]

I. Diener[‡]and R. Schaback[§]

Communicated by L.C.W. Dixon

1 Introduction

The problem of determining the global minimum and all global minimizers of a sufficiently smooth function f on a subset of $\mathbb{R}^n$ has received increasing interest during the last two decades. The most widely used methods are of stochastic nature and only comparatively few deterministic methods have been considered. Among the deterministic methods the continuous Newton method is of particular interest, since it is related to the well known usual Newton method and since the trajectories it generates have very nice properties. In section 1.1 we describe this method. In sections 1.2 and 1.3, we recall an extension first introduced in Refs. 1–2 which overcomes some shortcomings of the original method. In sections 2 and 3 we describe a numerical approach and some implementational details. Finally, in section 5, we present the results of some numerical tests of the method.

[†]The authors would like to thank L.C.W. Dixon for pointing out some errors in the original version of this paper and for several suggestions of improvements.

[‡]Assistant Professor, Institut for Numerical and Applied Mathematics, University of Göttingen, Göttingen, Germany.

[§]Full Professor, Institut for Numerical and Applied Mathematics, University of Göttingen, Göttingen, Germany.

1.1 Branin's Method

Let $f : B \to I\!R$ be a twice differentiable function on the bounded region $B \subset I\!R^n$. We consider the problem of finding *all* critical points of f in B by using a trajectory method. Such methods consist in numerically following certain implicitly defined curves in B which theoretically contain all critical points. One such method was proposed by Branin (Ref. 3) in 1972 and has since then become known as the *Continuous Newton Method.* Actually Branin considered the more general problem of determining all zeros of a function $F : I\!R^n \to I\!R^n$. In our setting we take F to be the gradient ∇f of f.

We denote the Hessian matrix of f at x by $H_f(x)$, its adjoint matrix by $\tilde{H}_f(x)$, and consider the following autonomous differential equation:

$$dx/dt = -\tilde{H}_f(x)\nabla f(x). \tag{1}$$

If the integration is started in x_0, the direction of $\nabla f(x)$ along the solution curves obviously stays parallel to $\nabla f(x_0)$. This important property is used later to define a more general set of curves. Euler's discretization yields

$$x_{k+1} = x_k - h_k\tilde{H}_f(x_k)\nabla f(x_k) \tag{2}$$

with suitably chosen steplengths h_k. If $|\tilde{H}_f(x)| \neq 0$ we have the relation $\tilde{H}_f(x) = |H_f(x)|H_f^{-1}(x)$ and (2) is just the damped Newton iteration method for finding zeros of ∇f, i.e. critical points of f.

Equation (1) has two types of stationary points. One occurs if $\nabla f(x) = 0$, i.e. if x is a critical point of f. The other occurs if $\nabla f(x) \neq 0$ but $\tilde{H}_f(x)\nabla f(x) = 0$. These were called *extraneous singularities* by Branin.

Branin's method consists in selecting a starting point x_0 and following the corresponding curve defined by (1). If a critical point is found on the path, it is output and the sign in (1) reversed. The integration is then continued until the next critical point is found or a termination criterion is satisfied.

The numerical implementations of Branin's algorithm so far have used different integration techniques, sometimes coupled with interpolation schemes (see Refs. 4–6). These approaches do not seem to be particularly efficient, since they hardly take the special structure of the defining equation (1) into account.

The main problem with this method (and all other trajectory methods) is, that in general not all critical points of f lie on a single connected trajectory component and thus other trajectory components have to be found. This problem was tackled in Ref. 2 by an extension of Branin's approach. The corresponding numerical method will be described in this paper.

1.2 Extension of Branin's Method

In Ref. 1 a new way to look at the trajectories generated by the continuous Newton method was proposed. The trajectories can be obtained in the form $(T_g f)^{-1}(0)$ for suitably defined functions $(T_g f) : \mathbb{R}^n \to \mathbb{R}^{n-1}$.

Definition1.1. Let $f : \mathbb{R}^n \to \mathbb{R}$ be continuously differentiable and let $g \in \mathbb{R}^n$ be a fixed nonzero direction. Then we call the set

$$T_g(f) := \{x \in \mathbb{R}^n | \, \nabla f(x) \text{ and } g \text{ are parallel}\}$$

a *Newton trajectory* of f. Let $G : \mathbb{R}^n \to \mathbb{R}^{n-1}$ be a linear map of rank $n-1$ such that $Gg = 0$. Except for this, G can be arbitrarily chosen, e.g. as $G = (g_1^t, g_2^t, \ldots, g_{n-1}^t)^t$, where $g_1, g_2, \ldots, g_{n-1}, g_n := g$ form an orthonormal basis of $\mathbb{R}^n$ and ^{t} stands for transposition. We use G later for dimensional recursion (see section 3.3). With a fixed choice of such a matrix G we define the continuous map $(T_g f)$ by

$$(T_g f) : \mathbb{R}^n \to \mathbb{R}^{n-1}, \qquad x \mapsto G \nabla f(x).$$

Obviously we have $(T_g f)^{-1}(0) = T_g(f)$, and

$$T_g(f) = \{x \in \mathbb{R}^n | G \nabla f(x) = 0\}. \tag{3}$$

The set $(T_g f)^{-1}(0)$ is independent of the special choice of G as long as the kernel of G is spanned by g. Also, $T_g(f) = T_h(f)$ if g and h are linearly dependent and nonzero. The (trivial) but important property of the Newton trajectories is given by the following

Theorem 1.1 Let $g, h \in \mathbb{R}^n$ be linearly independent vectors and f continuously differentiable. Then $T_g(f) \cap T_h(f) = \text{Crit}(f)$, the set of critical points of f.

Thus, in principle, one can find all critical points by completely tracing a Newton trajectory for f. In practice, however, various difficulties arise.

First, $T_g(f)$ might not be one–dimensional. This problem can be removed by imposing generic regularity conditions on f. A harder problem is that $T_g(f)$ is not connected in general. Conditions on f which guarantee connectedness were given in Ref. 1, but these conditions are too strong to apply in the general case.

An attempt to cope with this problem was made in Ref. 2. The construction described in that paper is very general and leads to a locally 1-dimensional net of curves connecting all critical points. However, numerical tracing of the whole net is not generally possible, because of certain "one–way–streets", i.e. paths whose "entrances" can be found, whose "exits", however, depend on global information about f and are overlooked by numerical methods. Details will be described below. Every critical point defines a numerically traceable subnet which often contains many or all critical points of f. In the next section we give a short summary of the construction for the special case of Newton trajectories . For details and the general construction the reader should consult Ref. 2.

1.3 Connecting the Components

Let $B \subset \mathbb{R}^n$ be a bounded convex region in R^n. Suppose $0 \neq g \in \mathbb{R}^n$ is given and the critical points of the function $f : B \to \mathbb{R}$ are distributed among several disconnected components of the locally 1-dimensional Newton trajectory $T_g(f)$. Let us further suppose that f is constant on the boundary of B and that this boundary contains no critical points of f. Define an auxiliary function Q on B by $x \mapsto g^t x$, where ^{t} stands for transposition and write $\hat{Q}(x)$ for $Q^{-1}(Q(x))$. Then it is shown in Ref. 2 that the set

$$T_g(f) \cup \bigcup_{x \in \Gamma} \hat{Q}(x) \tag{4}$$

is connected where the $\hat{Q}(x)$ are the *touching hyperplanes* of $T_g(f)$. Geometrically, (4) means that the disconnected components of $T_g(f)$ are joined together by a limited number of parallel hyperplanes orthogonal to g. Only those hyperplanes are needed which are "touching" $T_g(f)$ in some point x. These "touching points" form a set Γ. An example is given in Figure 3, and details are explained below. If $n = 2$ the set defined by (4) is already generically 1-dimensional and connected, since in this case the Q-contours are straight lines. If $n \geq 3$, the Q-contours are intersections of hyperplanes in $I\!R^n$ with the convex set B. However, the entire Q-contours are not needed. Only the (nonempty) set

$$T_g(f) \cap \bigcup_{x \in \Gamma} \hat{Q}(x) \tag{5}$$

is of interest since its elements are possible starting points for new components of $T_g(f)$.

In Ref. 2, it is shown that the set $T_g(f) \cap \hat{Q}(x)$ is equal to the set of critical points for the function f restricted to $\hat{Q}(x)$. Thus we have a problem of the same form as the original one, but posed on the $(n-1)$–dimensional linear space $\hat{Q}(x)$. This argument can be repeated, until the dimension of the base space is 2. Each problem generates locally 1-dimensional trajectories, and the overall result is a locally 1-dimensional trajectory net which contains $T_g(f)$. Recursion through dimensions involves a simplification, because more and more components of $G\nabla f(x)$ are replaced by linear functions (see section 3.3 below).

For the purpose of the present paper a simplified concept of touching points and touching hyperplanes suffices:

Definition1.2. Let x be a point on $T_g(f)$. Then x is called a *touching point* of $T_g(f)$ if there exists a neighbourhood U of x such that $T_g(f) \cap U$ is contained in only one of the two closed halfspaces defined by $\hat{Q}(x)$. The set $\hat{Q}(x)$ is then called a *touching hyperplane*. Let Γ be a subset of all touching points that contains at least one touching point from every touching hyperplane.

Let x be a point on $T_g(f)$ and suppose $T_g(f)$ is a C^1 curve in a neighborhood of x. Let $\phi : [0,1] \rightarrow B$ be a regular local parametrization of $T_g(f)$, and $x = \phi(t_0)$. Then $\hat{Q}(x)$ is a touching hyperplane, if t_0 is an isolated zero of $g^t\dot{\phi}(t)$ with sign change or a point of an interval where $g^t\dot{\phi}(t)$ vanishes identically. This fact will be used later as a criterion for the detection of touching hyperplanes.

2 Numerical Method

2.1 Overview

The special form (3) of the trajectories will in principle be invariant during recursion through the dimension of the problem. Therefore we propose a numerical method that makes exhaustive use of this special structure.
The algorithm will be described bottom–up in several stages:

(i) Computation of local steps;

(ii) Control of stepsize and direction;

(iii) Detection and treatment of exceptional points;

(iv) Recursion through dimensions;

(v) Bookkeeping strategy.

It uses a fixed vector $g \in I\!R^n \setminus \{0\}$ and an arbitrarily chosen matrix $G : I\!R^n \longrightarrow I\!R^{n-1}$ with $\text{rank} G = n-1$ and $Gg = 0$.

2.2 Computation of Local Steps

A local step at a point $x \in I\!R^n$ near a trajectory $T_g(f)$ will be dependent on a **single** control parameter $p \in]0, \frac{1}{2}dcptp]$, where *dcptp* is an input parameter describing the minimal expected distance between exceptional points, i.e. critical points. It steers the "resolution" of the algorithm. The local step $h(x,p)$ is computed under the assumption $\text{rank}(GH_f(x)) = n-1$ by

Step L1. Find a vector $z(x) \in I\!R^n$ with

$$\begin{aligned} GH_f(x)z(x) &= 0, \\ z^t(x)z(x) &= 1. \end{aligned}$$

This can be done by incomplete LU factorization, and the decomposition should be stored for later use. Since $z(x)$ is determined only up to a sign, we employ a fixed strategy to choose a sign, but allow an additional control parameter $\sigma \in \{-1, +1\}$ to reverse orientation of $z(x)$.

Step L2. Solve the system

$$\begin{pmatrix} GH_f(x) \\ z^t(x) \end{pmatrix} h(x,p) = \begin{pmatrix} -G\nabla f(x) \\ p \end{pmatrix}$$

for the step $h(x,p)$, using the factorization calculated in step L1.

We note some simple facts:

Theorem 2.1. The step $h(x,p)$ satisfies

(a) $h(x,p) = h(x,0) + p \cdot z(x)$;

(b) $z(x)$ is tangential to $T_g(f)$ if $x \in T_g(f)$;

(c) $h(x,0)$ is the Newton step for the system $G\nabla f(x) = 0$ defining $T_g(f)$;

(d) $z(x)$ is orthogonal to $h(x,0)$.

If x is on the trajectory, properties a) and b) imply that $h(x,p)$ is a predictor step for large p, where p acts as a stepsize. Properties a) and c) show that $h(x,p)$ is a corrector for small p, performing a Newton step towards the trajectory. Indeed, when x is near the trajectory $T_g(f)$, the vector $z(x)$ is nearly parallel to $T_g(f)$, and since $h(x,0)$ is orthogonal to $z(x)$, it will point towards the trajectory, being approximately orthogonal to the tangent at the nearest point (see Figure 1).

For efficiency reasons, property a) suggests to compute $h(x,0)$ first in step L2; then control of p does not require much additional work for computing $h(x,p)$ for different values of p.

In contrast to other methods (see Refs. 10–11) we only need a single parameter to control both predictor stepsize and the relative weight of prediction versus correction. The drawback is that we do not use high–order predictors.

2.3 Control of Local Steps

The choice of the parameter p of a local step should make sure that the progress along the trajectory is as large as possible under the restriction that the trajectory is not “lost”. To define the latter notion we simply postulate that there exists a neighborhood U of $x + h(x,p)$ such that a unique point of

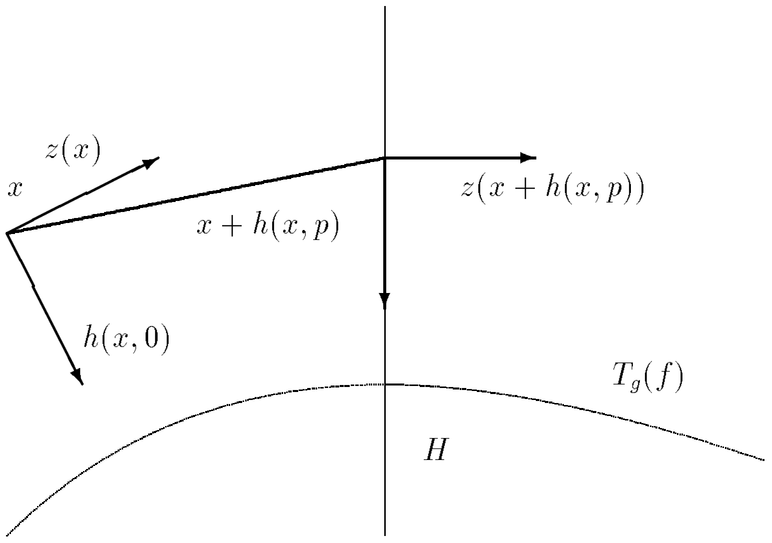

Figure 1: Local control.

the trajectory should exist in $U \cap H$ for a hyperplane H through $x + h(x,p)$ with normal vector $z(x + h(x,p))$ (see Fig. 1).

This can be guaranteed by the Newton–Kantorovitch theorem (see e.g. Ref. 7) applied to the system

$$A_{x,p}(y) := \begin{pmatrix} G\nabla f(x + h(x,p) + y) \\ z^t(x + h(x,p))y \end{pmatrix} = 0,$$

because for a solution $Y(x,p)$ of $A_{x,p}(Y(x,p)) = 0$ we have

$$\begin{array}{rcl} x + h(x,p) + Y(x,p) & \in & T_g(f), \\ z^t(x + h(x,p))Y(x,p) & = & 0. \end{array}$$

The standard form of the Newton–Kantorovitch theorem explicitly gives two radii $0 < r < R$ of balls B_r and B_R around the starting point $y = 0$; there is a solution of the system in the smaller ball and there is no other solution in the larger ball. In our case the required bounds for

$$\begin{array}{rcl} \|(A'_{x,p})^{-1}(0)\| & \leq & D, \\ \|(A'_{x,p})^{-1}(0)A_{x,p}(0)\| & \leq & \eta \end{array}$$

can be calculated directly, and the Lipschitz constant L for $A'_{x,p}(y)$ as a function of y can be estimated numerically. We then **accept** a step $h(x,p)$, if the Newton–Kantorovitch theorem is applicable, i.e. if $2D\eta L \le 1$. This guarantees existence of the solution $Y(x,p)$ within

$$\|Y(x,p)\| \le D^{-1}L^{-1}(1-\sqrt{1-2DL\eta}) =: r$$

and uniqueness within

$$\|Y(x,p)\| \le D^{-1}L^{-1}(1+\sqrt{1-2DL\eta}) =: R,$$

if the Lipschitz constant L is assumed to be correctly estimated. The actual radii r and R for existence and uniqueness are not explicitly controlled by our method. It suffices to be sure that there always is a unique trajectory point near to the actually calculated point. The flaws of this argument are that the Lipschitz constant is only an estimate and that acceptance of $h(x,p)$ tacitly assumes that all smaller values of p are acceptable, too. This is motivated by Theorem 2.1, but leaves a possibility for the algorithm to switch to a different locally unique trajectory without notice. This, however, is an improbable event because switching over to a hidden quasi–bifurcation branch (see Fig. 2) will often give a jump in the orientation of $z(x)$. This jump is detectable by checking the sign of $z^t(x)z(x+h(x,p))$, using the fact that the trajectory is traced with a fixed algorithm determining the local orientation of $z(x)$ in continous dependence on x.

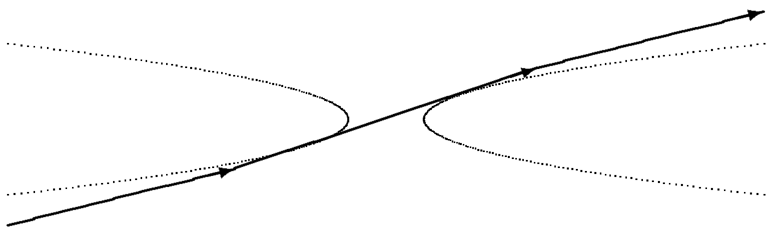

Figure 2: Quasi–bifurcation with hidden switch to another trajectory component, recognizable by checking orientation.

Control of p is simply done by decreasing p by a certain factor less than unity whenever $h(x,p)$ is not accepted. This brings the new direction "closer" to the actual trajectory. After an accepted step, p is increased again.

3 Exceptional Cases

3.1 Regularity and Bifurcation

Since $T_g(f) = (G\nabla f)^{-1}(0)$ the assumption rank$GH_f = n-1$ on $T_g(f)$ is **generic**, because 0 needs only to be a regular value of $G\nabla f$ to ensure non-degeneracy. Small perturbations of g will remove singularities, and in practice the algorithm will encounter only "quasi–bifurcations" in the sense of Fig. 2.

The mildly degenerate cases of this sort are overcome by the automatic use of small values of p along a single trajectory; the others will lead to a rank loss when trying an incomplete LU factorization in step L1. If the rank loss concerns just one dimension, a projection onto a two–dimensional subspace is possible, and the asymptotes of hyperbolae like in Fig. 2 can be calculated giving the necessary information to handle the bifurcation. Other cases have to be eliminated by choosing other values of g, such that (hopefully) 0 is now a regular value for $G\nabla f$. Practical experience shows that it is not worthwhile to handle genuine bifurcations explicitly, because they are very rare indeed and the necessary computing effort is better invested into tracing a trajectory system for a different g. We therefore simply assume for our method that 0 is a regular value for $G\nabla f$.

3.2 Critical Points and Touching Points

In contrast to earlier trajectory algorithms the tracing method described so far will only slow down near quasi–bifurcations or in regions with very large Lipschitz constants for the Hessian along the trajectory, but not necessarily near critical points or touching points. These are detected by additionally monitoring two simple real–valued functions that necessarily vanish at these points and have a sign change in nondegenerate cases:

$$\begin{array}{rcll} \gamma_g(x) & := & g^t\nabla f(x), & \text{for critical points,} \\ \theta_g(x) & := & g^t z(x), & \text{for touching points.} \end{array}$$

Theorem 3.1.

(a) The function $\gamma_g(x)$ has zeros on $T_g(f)$ in critical points only, and these zeros x^* are simple, if $g \notin \ker H_f(x^*)$, e.g. if $H_f(x^*)$ is nondegenerate.

(b) Isolated touching points $\tilde{x}$ on $T_g(f)$ are characterized by the property that $\theta_g(x)$ has a zero with sign change at $\tilde{x}$.

(c) If $T_g(f)$ is nondegenerate around an isolated touching point, $\theta_g(x)$ will have a zero with sign change on any curve sufficiently close to $T_g(f)$.

Proof : The equations

$$g^t \nabla f(x) = 0 \quad \text{and} \quad G\nabla f(x) = 0$$

imply $\nabla f(x) = 0$ and vice versa; a zero x^* of the derivative of γ_g would imply

$$\nabla \gamma_g(x^*) = g^t H_f(x^*) = 0,$$

proving assertion a). The other statements are immediate consequences of the definition of touching points. □

If $H_f(x^*)$ is nonsingular in a critical point x^*, the trajectory $T_g(f)$ intersects the zero set of $\gamma_g(x)$ transversally, and therefore the numerical detection of critical points via a sign change of $\gamma_g(x)$ is numerically stable even if the trajectory is not followed exactly.

The exact detection of touching points is somewhat more problematic, because their definition already involves a higher derivative than needed for critical points. Fortunately, touching points need not be calculated very precisely for the following reasons:

(i) Imprecisely calculated touching points can be seen as exact touching points of trajectories with a slightly perturbed g. The algorithm follows a G–contour on a hyperplane parallel and very near to the exact one, until a new trajectory component is hit (see Fig. 3 for illustration). In nondegenerate situations this trajectory component will cut all parallel hyperplanes transversally, and the algorithm simply reaches the new trajectory component in s slightly different point. Here the "one–way–streets" come up again: touching points are the well-defined "entrances", but the exits lie somewhere on a new trajectory component. Errors in the calculation of touching–points just result in tracing a neighbouring "one–way–street", but the exit will normally lead to the **same** new trajectory component.

(ii) If a touching point is falsely assumed to exist, the algorithm gets an additional chance for recursion, and no information is lost.

The actual calculation of critical points and touching points employs the following strategy:

Step C1. Halt trajectory tracing whenever one of the indicator functions γ_g or θ_g is very small or has a sign change.

Step C2. Try to find an approximate zero or a local minimum of the indicator function by bisection. Use small values of p for this calculation, to make sure to be very near the actual trajectory $T_g(f)$.

Step C3. Record the solution for later use, and follow the bookkeeping strategy described in section 3.4.

3.3 Dimensional Recursion by Projection

If

$$G := \begin{pmatrix} g_1^t \\ g_2^t \\ \vdots \\ g_{n-1}^t \end{pmatrix}, \quad g_i \in \mathbb{R}^n, \ \ 1 \le i \le n, \ \ g_n := g$$

is the matrix chosen when starting the procedure for the original n–dimensional problem, dimensional recursion simply uses the system

$$\begin{pmatrix} g_1^t H_f(x) \\ \cdot \\ \cdot \\ \cdot \\ g_{m-1}^t H_f(x) \\ g_{m+1}^t \\ \cdot \\ \cdot \\ \cdot \\ g_n^t \\ z^t(x) \end{pmatrix} h(x,p) = \begin{pmatrix} -g_1^t \nabla f(x) \\ \cdot \\ \cdot \\ \cdot \\ -g_{m-1}^t \nabla f(x) \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ p \end{pmatrix},$$

where $z(x)$ is a normalized solution of the homogeneous system without the last equation. Given a starting point $x \in \mathbb{R}^n$ for the m–dimensional subproblem, everything is projected onto the affine space $x + \operatorname{span}\{g_1, \ldots, g_m\}$, and after some simple reductions eliminating $n-m$ homogeneous equations the problem has the same form as before. The reductions are independent of x and can be done once for all before actually tracing the trajectory.

3.4 Bookkeeping Strategy

The algorithm normally detects many critical points and touching points for subproblems on different dimension levels, forming a tree–like dynamical data structure. To avoid multiple tracing of the same or neighboring trajectories, and to avoid low–dimensional subproblems whenever possible, a serious bookkeeping problem arises which greatly influences the performance of the algorithm.

The algorithm has been programmed by A. Drexler (Ref. 8) who solved most of the implementational problems. To keep a record of what is achieved, a dynamically varying list of possible and yet unused starting points and of critical points on each level is kept. On each dimension level m the traced trajectories start in a touching point $\tilde{x}_{m+1}$ which was found and registered on level $m+1$. Therefore the starting point of the whole algorithm is formally classified and recorded as a touching point of a virtual problem of dimension $n+1$. The trajectory on level m has to be traced in both directions (this is where the initial orientation of $z(\tilde{x}_{m+1})$ has to be chosen and recorded in order to be reversed later) and a backward pointer to $\tilde{x}_{m+1}$ is kept for the whole tracing.

The computation halts at the boundary of a prescribed large cube B, to which the whole process is assumed to be confined, or if a critical– or touching point is found twice on level m. The latter is done to recognize cyclic trajectories and to avoid repeated tracing of parts of the current trajectory.

In either case the local trajectory section is declared to be completely traced; if this is the case for the other orientation, too, the complete trajectory component is considered to be traced, and $\tilde{x}_{m+1}$ is marked as "traced" on the list.

New critical– or touching points found on level m are added to the list on that level. For efficiency reasons, tracing continues until a trajectory–component on level m has been traced in both directions.

To start a new trajectory the tree–like data structure is searched for available new starting points, always beginning at the highest possible dimension level. The search algorithm uses the fact that a critical point recorded on level m starts a new trajectory component on level $m+1$ while a touching point recorded on level m starts a new trajectory on level $m-1$. Recording the exceptional points on “their” level facilitates cycle checking, which must be efficient because it occurs very frequently.

3.5 Reducing the Number of Touching Points

Whenever the quantity

$$\tilde{\rho} := \left| g_m^t \tilde{x}_m^1 - g_m^t \tilde{x}_m^2 \right| \tag{6}$$

for two touching points $\tilde{x}_1$ and $\tilde{x}_2$ on level m on the same trajectory is small, the new trajectory components T_1 and T_2 starting at $\tilde{x}_1$ and $\tilde{x}_2$ on level $m-1$ will be confined to parallel touching hyperplanes on level m with distance $\tilde{\rho}$. On both hyperplanes the same function is used to define the trajectory, and thus T_1 and T_2 will differ only slightly on level $m-1$. This also holds for the critical points on them, and these will normally lead then to the **same** trajectory on level m again. Since we are really interested in touching hyperplanes and not in touching points, it is reasonable to start the recursion in only one of $\tilde{x}_m^1$ and $\tilde{x}_m^2$. Note that this argument is quite similar to the one used to explain possible tolerances in the calculation of touching points.

Thus each new touching point $\tilde{x}_m$ of a trajectory on level m is compared to the other touching points already found on the same level. If the difference defined by (6) is less than an input parameter ρ, the touching point $\tilde{x}_m$ is discarded. This important additional strategy strategy keeps the algorithm from tracing large numbers of similar trajectories on parallel touching hyperplanes without getting any new information (see fig. 3). It is not sensitively dependent on the actual value of ρ, provided that ρ is neither zero nor extremely small (see Table 2 below).

4 Example

In order clarify some of the points made in the preceding sections we now discuss a larger example. It has been produced by applying the present method to the function f_a defined in section 5, where $n := 4$ and $d := 5$. The parameter ρ was set to 1 and we considered the region $[-6,6]^4 \subset I\!R^4$. Furthermore, we traced three levels, i.e. we had to solve subproblems of dimension 4,3 and 2. Fig. 4 shows the graph traced by the Algorithm.

The three levels in the figure correspond to the 4,3 and 2 dimensional subproblems. The different trajectory components are represented by strings consisting of the symbols $T, t, C, c, \emptyset$ and $\exists$. T and t stand for touching points and C and c for critical points on the respective level (dimension). The symbol $\emptyset$ denotes a trajectory on which no touching points and no critical points were found. The symbol $\exists$ means that a trajectory was found, which had been traced before and thus is not traced again. In Fig. 4 the trajectory components on the different levels are joined by arrows labeled by sequence numbers indicating the order in which the tree was traversed.

One type of arrow originates at a touching point (denoted by the symbol 't') at level $L = 4$ or 3 and leads to a subproblem in dimension $L - 1$. Thus such an arrow points to a trajectory component at level $L - 1$. The other type of arrow originates at critical points on a level $L = 2$ or 3 and points to a trajectory component traced at level $L + 1$. The symbols 'T' denote touching points which were not expanded because of the ρ-heuristic described in section 3.5, or because they occured on the lowest possible level, i.e. on level 2. The symbols 'c' denote critical points of an L-dimensional subproblem which initiates a new problem in dimension $L + 1$. The symbols 'C' on level L stand for critical points that do not initiate a new problem on level $L + 1$ either because $L = 4$ or they result from the fact that a touching point t on level L is a critical point for the corresponding subproblem on level $L - 1$.

As can be seen from figure 4 the algorithm found 9 critical points on level 4. These are critical points of f_a.

The algorithm was started at the origin and the vector g was taken as the gradient of f_a at the origin. The first trajectory component traced contained 2 critical points of f_a (arrow labeled 1). Furthermore, two touching points were found.

Then a 3-dimensional problem was started at the first touching point found on level 4 (arrow 2). This 3-dimensional problem led to 2 touching points on level 3. One of them was discarded by the ρ-heuristic. It is thus marked with 'T'. Next, according to the strategy described in section 3.4, a new 3-dimensional problem is started at the second touching point found on level 4 (arrow 3). This leads to 2 new touching points on level 3.

To continue, the algorithm selected the touching point remaining in the first trajectory traced on level 3 (arrow 4) and starts a subproblem on level 2, i.e. a 2-dimensional problem. Two touching points and one critical point are found. The two touching points are labeled 'T' rather than 't' since the bottom level was reached and no dimensional recursion was to be performed.

At the next step the algorithem selects the critical point found on level 2 and starts in it a new problem on level 3 (arrow 5). This leads to 3 critical points on level 3 which are expanded (and lead to problems on level 4) into the trajectories pointed to by the arrows labeled 6,7 and 8. One of the new trajectories on level 4 contains no critical points and no touching points. Thus it is labeled $\emptyset$ in figure 4. The other two trajectory components on level 4 contain respectively 2 and 5 critical points of f_a and no further critical points are found by subsequent tracing.

5 Numerical Tests

5.1 Test Problems

The algorithm has been tested on a large number of problems with up to 32 dimensions. Along with some of these problems we include here, for the purpose of comparison, some standard test problems extracted from the literature.

1. The two–dimensional *six–hump–camelback* function defined by

$$f_c(x) := \frac{1}{3}x_1^6 - 2.1x_1^4 + 4x_1^2 + x_1x_2 - 4x_2^2 + 4x_2^4$$

   is a standard test problem. The function f_c has 15 critical points in the region $[-2.5, 2.5]^2$. Their minimum distance from each other is greater than 0.3.

2. This test problem is the error function of a two–dimensional linear Čhebyšhev approximation:

$$\begin{aligned} f_t(x) &:= 0.066581 \sin(\pi x_1) \sin(\pi x_2) \\ &+ 0.002503 \big(\sin(\pi x_1) \sin(3\pi x_2) + \sin(3\pi x_1) \sin(\pi x_2)\big) \\ &+ 0.000086 \sin(3\pi x_1) \sin(3\pi x_2) \\ &+ 0.000559 \big(\sin(\pi x_1) \sin(5\pi x_2) + \sin(5\pi x_1) \sin(\pi x_2)\big) \\ &- x_1(1-x_1)x_2(1-x_2). \end{aligned}$$

The function f_t has 51 critical points in the region $[0,1]^2$. Their minimum distance from each other is greater than 0.05 . Many of the critical points have narrow oval regions of attraction.

3. This problem is defined by

$$f_e(x) := \sum_{i=1}^{d} \alpha_i e^{\lambda_i\left((x_1-x_1^i)^2+(x_2-x_2^i)^2\right)}$$

where $d = 5$ and

$$\begin{array}{llll} \alpha_1 = 2, & \lambda_1 = -1, & x_1^1 = -2, & x_2^1 = 0, \\ \alpha_2 = 3, & \lambda_2 = -2, & x_1^2 = 3, & x_2^2 = 0, \\ \alpha_3 = 1, & \lambda_3 = -3, & x_1^3 = 1, & x_2^3 = 2, \\ \alpha_4 = 4, & \lambda_4 = -3, & x_1^4 = 0, & x_2^4 = 2, \\ \alpha_5 = 2, & \lambda_5 = -2, & x_1^5 = 0, & x_2^5 = -1. \end{array} .$$

The function f_e has 9 critical points in the region $[-5,5]^2$. Their minimum distance from each other is greater than 0.95 . The maxima of this function are sharp peaks. A contour plot of this function along with a Newton–trajectory and the touching hyperplanes is shown in Fig. 3.

4. The four–dimensional *Shekel* functions are defined by (cf. Ref. 9)

$$f_s(x) := -\sum_{i=1}^{d} \frac{1}{\|x-a^i\|_2^2 + c_i},$$

where $d \geq 1$, the a^i are vectors in $I\!R^n$, and the c_i are real numbers defined as in Ref. 9. Common values for d are 5, 7, and 10. (In the literature the resulting functions functions are usually denoted by SQRN5, SQRN7 and SQRN10.) For these d the functions f_s apparently have 11, 13, and 21 critical points in the region $[0,12]^4$. Their minimum distance from each other is greater than 0.55, 0.9, and 0.75 . Some of the extrema of f_s have tiny regions of attraction.

5. The last example is n–dimensional:

$$f_a(x) := \sum_{i=1}^{d} \arctan\left(\|x-a^i\|^2\right)$$

on the region $[-6,6]^n$, where $d \geq 1$ and the vectors $a^i \in [-6,6]^n$ are randomly chosen. The number of critical points and the appropriate domain of search both depend on d and the a^i. For $d=3$ and $d=10$ and the vectors chosen for the examples given in Tab. 4 the number of critical points is given by the righthand entries of the table below and the minimum distance of critical points from each other is given by the lefthand entries:

n	$d=3$		$d=10$	
2	2.0	5	0.2	5
8	1.7	5	1.6	13
16	3.5	5	1.5	13

5.2 Test Results

The tests were conducted by A. Drexler using the program described in Ref. 8. Each test was run 10 times with random starting points x_s and fixed estimates for *dcptp* and ρ. The vector g was computed as $g := \nabla f(x_s)$, and the maximal number of levels to be traced was prescribed via a constant l_{max}.

In Tables 1–4, the *len* entries list the average length of the trajectories by levels and in total. The lengths were computed by seperately summing the lengths of the steps taken in the various $(n-k)$–dimensional subproblems for $k = 0, 1, \ldots, l_{max} - 1$. The *evals* entries list the average number of gradient– and Hessian evaluations by levels and in total. Since the ultimate goal was to find all critical points and numerous critical points were found in every case the *succ* entries list the percentage of runs that successfully found **all** critical points.

The "Level" entry indicates the dimension of the subproblem traced in the corresponding row; when there are l_{max} traced levels for an n–dimensional problem, we have l_{max} rows with dimensions $n, n-1, \ldots, n-l_{max}+1$ indicated below "Level".

5.3 Conclusions

The test results show that the method is able to find **all** critical points for an astonishingly large percentage of test runs. It should not be applied with high resolution (i.e. *dcptp* small, $\rho = 0, l_{max} = n$) for a single value of g and x_s, tracing trajectories recursively through all levels, because the computing

Table 1: Results for the functions f_c and f_t.

		f_c, $dcptp = 0.3$, $\rho = 0$			f_t, $dcptp = 0.05$, $\rho = 0$		
$n = 2$	Level	*len*	*evals*	*succ*	*len*	*evals*	*succ*
$l_{max} = 1$	2	15.0	465	100%	7.8	1536	90%
$l_{max} = 2$	2	16.5	533		10.0	2131	
	1	13.9	112		4.7	294	
	Total	30.4	645	100%	14.7	2425	100%

Table 2: Results for the function f_e.

$dcptp = 0.95$		$\rho = 0$			$\rho = 0.9$			$\rho = 1000$		
$n = 2$	Level	*len*	*evals*	*succ*	*len*	*evals*	*succ*	*len*	*evals*	*succ*
$l_{max} = 1$	2	17.3	482	20%						
$l_{max} = 2$	2	52.2	1671		46.0	1425		33.9	1015	
	1	62.9	211		42.7	142		11.0	38	
	Total	115	1882	90%	88.7	1567	90%	44.9	1053	80%

effort increases too much (see Table 4, $n = 8, 16$). However, the method has rather good chances to find all critical points when applied for a small number of different choices of x_s and g, tracing only a small number of levels with values of *dcptp* and ρ which are not too small. Recursion by at least one or two levels is strongly recommended (see Table 3). The chance of finding all critical points is enhanced, of course, when several runs with different starting points are made.

It is always a difficult issue to compare the efficiency of different algorithms for global optimization. For local optimization one common criterion is the local convergence rate and the size of the regions of convergence. But for global optimization there is, as yet, no generally accepted measure for the efficiency on which comparisons could be based. One usually compares the number of function evaluations for some test problems. Furthermore, our method is designed to compute all stationary points and not just the global minimizers. Since many methods were tested on the Shekel–functions f_s and results for this function are given in the literature we shall briefly compare our results for these functions (Table 3) with the results obtained by other researchers (cf. Ref. 9).

Branins original method needs 5500, 5020 and 4860 gradient and hessian

Table 3: Results for the functions f_s.

		$d=5$, $dcptp=0.55$			$d=7$, $dcptp=0.9$			$d=10$, $dcptp=0.75$		
$n=4$	Level	*len*	*evals*	*succ*	*len*	*evals*	*succ*	*len*	*evals*	*succ*
$l_{max}=1$	4	28.3	551	0%	37.9	715	0%	29.7	540	0%
$l_{max}=2$	4	46.0	942		64.0	1233		68.7	1304	
$\rho=1.0$	3	61.5	1153		79.0	1388		59.1	987	
	Total	108	2095	30%	143	2661	30%	128	2291	10%
$l_{max}=4$	4	77.8	1646		94.2	1823		125	2419	
$\rho=1.0$	3	170	3277		185	3190		187	3051	
	2	176	1806		214	1805		235	2325	
	1	29.9	121		44.3	120		85.8	279	
	Total	454	6850	100%	537	6938	100%	633	8074	60%

evaluations for SQRN5, SQRN7 and SQRN10 respectively. In a much more efficient implementation of the method by J. Gomulka (cf. Ref. 5) these numbers were reduced to 275, 251 and 243 gradient and hessian evaluations respectively. Since these implementations traced only one component on level n we have to compare these results with the numbers given in the first row of table 3. Thus the numbers for our method are 551, 715 and 540 gradient and hessian evaluations. In comparing these numbers one has to keep in mind that our results are *averages* obtained by starting the method in 10 randomly generated starting points within the feasible region whereas the numbers given by Gomulka are the results for a single run. So we might claim that the efficiency of our implementation (considered as an implementation of Branins method, i.e. with $l_{max}=1$) is roughly comparable to the efficient implementation by Gomulka. It has been already observed by Gomulka that, for these examples, it is a matter of "pure chance", wether the ordinary Branin method finds the global minimizer. However, as can be seen from the results in table 3 for $l_{max}=2,4$, our scheme of dimensional recursion greatly enhances the probability that *all* stationary points are found. For $l_{max}=2$ and 4 the global minimizer was found for all three functions in every case.

A comparison of our method with a stochastic method is even more difficult. We shall briefly compare the present method with the efficient "Multi Level Single Linkage" (MLSL) method proposed in Ref. 12. The authors report 404, 432 and 564 function evaluations respectively. However, it is not clear from their presentation, how these numbers were obtained. For

Table 4: Results for the functions f_a.

		$d = 3$			$d = 10$		
	Level	*len*	*evals*	*succ*	*len*	*evals*	*succ*
$n = 2$			$dcptp = 2.0$			$dcptp = 0.2$	
$l_{max} = 1$	2	21.6	263	60%	14.2	357	0%
$l_{max} = 2$	2	30.6	360		23.5	626	
$\rho = 1.0$	1	32.3	49		29.8	316	
	Total	62.9	409	100%	53.4	942	30%
$n = 8$			$dcptp = 1.7$			$dcptp = 1.6$	
$l_{max} = 1$	8	36.8	1013	60%	37.5	985	0%
$l_{max} = 2$	8	59.8	1643		69.5	1818	
$\rho = 1.0$	7	65.1	1770		60.8	1644	
	Total	125	3413	80%	130	3462	10%
$n = 16$			$dcptp = 3.5$			$dcptp = 1.5$	
$l_{max} = 1$	16	30.8	559	0%	85.0	3068	10%
$l_{max} = 2$	16	77.6	1392		147	5358	
$\rho = 1.0$	15	51.8	1055		117	4740	
	Total	129	2447	80%	264	10098	40%

instance, MLSL generates an initial sample of function values, then selects certain points from this sample in which local searches are started. For these local minimizations a variable metric method (VA10AD) was used. Thereafter the sample is increased and new local searches are initiated until a stopping criterium is satisfied. Obviously, at least the gradient of the function must also be evaluated a certain number of times. Also it is not clear, wether the function evaluations used to generate the initial sample were included in the numbers. MLSL did not find the global minimizer for SQRN7 in one of four runs. Although this problem could be avoided, by choosing different parameters for the stopping criterium, this modification resulted in more function evaluations (factors 2 and 3 for SQRN7 and SQRN10).

The efficiency of the numerical implementation described above can be further improved, for instance by using update schemes for the Hessian matrix. We do not claim that our method is more efficient than most other methods. Instead we feel that, in the lack of generally accepted measures for the performance of algorithms for global optimization, at present all methods have distinct advantages and drawbacks. The main purpose of this study was to show that the general idea of recursive descent proposed in Ref. 2 is numerically feasible and yields satisfactory results. Since Newton–trajectories have strong theoretical properties it can be hoped that future results on the topology of the Newton–trajectories can in some practically important cases guarantee that all critical points are found.

The idea of Newton–trajectories is not limited to $\mathbb{R}^n$ but can be extended to functions on differentiable manifolds. Moreover, insight into the geometric structure of the trajectories might also help with the study of the convergence behavior of the usual Newton method.

References

[1] Diener, I., *On the Global Convergence of Path-Following Methods to Determine All Solutions to a System of Nonlinear Equations*, Mathematical Programming, Vol. 39, pp. 181–188, 1987.

[2] Diener, I., *Trajectory Nets Connecting All Critical Points of a Smooth Function*, Mathematical Programming, Vol. 36, pp. 340–352, 1986.

[3] Branin, F.H., *A Widely Convergent Method for Finding Multiple Solutions of Simultaneous Nonlinear Equations*, IBM. Journal of Research and Development, pp. 504–522, 1972.

[4] Gomulka, J., *Remarks on Branin's Method for Solving Nonlinear Equations*, Towards Global Optimisation, Edited by L.C.W. Dixon and G.P. Szegö, North–Holland, Amsterdam, Holland, Vol. 1, pp. 96–106, 1975.

[5] Gomulka, J., *Two Implementations of Branin's Method: Numerical Experience*, Towards Global Optimisation, Edited by L.C.W. Dixon and G.P. Szegö, North–Holland, Amsterdam, Holland, Vol. 2, pp. 151–163, 1978.

[6] Hardy, J.W., *An Implemented Extension of Branin's Method*, Towards Global Optimisation, Edited by L.C.W. Dixon and G.P. Szegö, North–Holland, Amsterdam, Holland, Vol. 1, pp. 117–139, 1975.

[7] Ortega, J.M., and Rheinboldt, W.C., *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, New York, 1970.

[8] Drexler, A., *Zur Numerik eines erweiterten kontinuierlichen Newton–Verfahrens*, Universität Göttingen, Diplomarbeit, 1988.

[9] Dixon, L.C.W., and Szegö, G.P., *The Global Optimization Problem: An Introduction*, Towards Global Optimisation, Edited by L.C.W. Dixon and G.P. Szegö, North–Holland, Amsterdam, Holland, Vol. 2, pp. 1–15, 1978.

[10] Allgower, E. L., and Georg, K., *Predictor–Corrector and Simplicial Methods for Approximating Fixed Points and Zero Points of Nonlinear Mappings*, Mathematical Programming, Edited by A. Bachem, M. Grötschel and B. Korte, Springer–Verlag, Berlin, Germany, pp. 15–56, 1983.

[11] Georg, K., *Zur Numerischen Realisierung von Kontinuitätsmethoden mit Prädiktor–Korrektor- oder simplizialen Verfahren*, Habilitationsschrift, Bonn, Germany, 1982.

[12] Rinnooy Kan, A.H.G., Boender, G.C.E., and Timmer, G.T., *A Stochastic Approach to Global Optimization*, Report No. 8429/O, Erasmus University, Rotterdam, Holland, 1984.

List of Figures

List of Tables

Figure 3: Contour plot of the function f_e with a Newton trajectory and the touching hyperplanes.

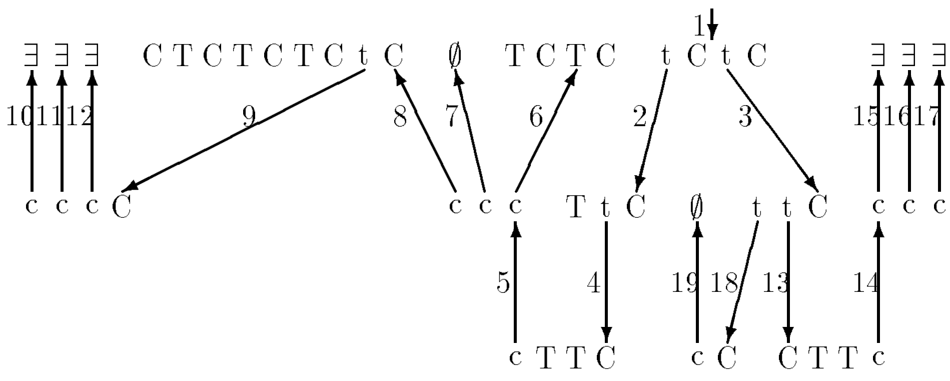

Figure 4: Graph traced by the algorithm. The function is f_a with $n = 4, d = 5$ and $\rho = 1$ in the region $[-6, 6]^4$.