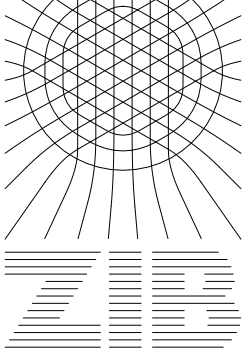
Konrad-Zuse-Zentrum für Informationstechnik Berlin
Heilbronner Str. 10, D-10711 Berlin - Wilmersdorf



Andreas Hohmann

Object Oriented Design of Multilevel Newton and Continuation Methods

SC-94-4 (March 1994)

OBJECT ORIENTED DESIGN OF MULTILEVEL NEWTON AND CONTINUATION METHODS

ANDREAS HOHMANN*

Abstract. We present a set of C++ classes that realize an abstract inexact Gauss Newton method in combination with a continuation process for the solution of parameter dependent nonlinear problems. The object oriented approach allows the continuation of different types of solutions within the same framework.

Key Words. inexact Gauss Newton method, tangential continuation, steplength control, parameter dependent nonlinear equations, dynamical systems, object oriented implementation, C++ class library

Introduction. It seems beyond doubt that Newton's method is the method of choice for the numerical solution of nonlinear problems provided with some differentiability. Combined with some (typically tangential) continuation process it also allows the numerical investigation of parameter dependent problems. Because its idea of successive linearization is so fundamental, this Euler-Newton process may be applied to rather different kinds of nonlinear problems. We only refer to the numerical analysis of (finite dimensional) dynamical systems regarding fixed points, periodic solutions, all sorts of bifurcations, and even connecting orbits. Here, the nonlinear problems are partly finite dimensional (fixed points and bifurcations of stationary solutions) and partly infinite dimensional (periodic and connecting orbits) which have to be discretized in some way. Despite the different nature of the nonlinear problems and their solutions, the basic numerical approach remains the same.

Naturally, this situation calls for an object oriented realization of the abstract numerical method. We present a set of C++ classes which realizes an inexact Gauss Newton continuation method for (almost) arbitrary parameter dependent nonlinear problems including infinite dimensional problems with varying discretizations. The crucial point is a carefully designed interface for the abstract inexact Newton method via virtual member functions. The structure of the continuation classes allows the continuation of different kinds of solutions (including bifurcations) within the same framework.

As an application, we consider parameter dependent dynamical systems and analyze the typical situation of a Hopf bifurcation. Here, we first have to follow a branch of stationary points (finite dimensional problem), detect and compute the Hopf bifurcation, and then follow the emanating branch of periodic solutions (infinite dimensional problem). The latter are computed via an adaptive multilevel Newton h-p collocation method using variable (adaptively chosen) grids for the discretizations of the periodic solutions.

With the abstract continuation classes at hand, it makes no difference whether we follow a branch of stationary or periodic solutions. The inexact Newton process as well as the adaptive steplength control for the tangential continuation apply in exactly the same way. The classes demonstrate that the object oriented design of advanced numerical methods allows an almost 1-1 representation of the abstract mathematical

* Konrad–Zuse–Zentrum Berlin, Heilbronner Str. 10, D-10711 Berlin-Wilmersdorf, Germany. hohmann@sc.zib-berlin.de

setting and thus simplifies the implementation drastically.

The paper is divided into two parts, the first devoted to the abstract mathematical setting and the second to its object oriented implementation. In the first section we also provide the numerical methods for the solution of underdetermined nonlinear problems: an inexact Gauss Newton method based on an affine contravariant convergence result and a simple control mechanism for the predictor stepsize in a continuation method.

The classes are part of a larger package for numerical computations, in particular for ordinary differential equations, which is available via anonymous ftp from the directory pub/code++ at elib.ZIB-Berlin.de (130.73.108.11). Here, the reader may also find a more detailed documentation [Hoh93a] of these classes.

1. Mathematical Setting and Numerical Methods. In what follows, X, Y, and Z denote Banach spaces and $D \subset X$ an open convex subset of X. We consider a parameter dependent nonlinear problem

$$F(z, \lambda) = 0 \,, \tag{1}$$

where $F : D \subset Z \times \Lambda \longrightarrow Y$ is a differentiable mapping, and Λ is some finite dimensional parameter space, $\dim \Lambda = p$. For convenience, we consider instead of (1) the underdetermined nonlinear problem in $X = Z \times \Lambda$ (cf. [DFK87])

$$F(x) = 0, \quad \text{where} \quad F : D \subset X \longrightarrow Y \,. \tag{2}$$

In order to obtain a well defined p-dimensional solution manifold $\mathcal{M} \subset \mathcal{D}$, we have to require (see e.g. Fink and Rheinboldt [FR83]) that F is a Fredholm mapping of index p with $\dim \ker F'(x) = p$ for all solutions $x \in D$, $F(x) = 0$. Thus, $0 \in Y$ is a regular value of F and (2) defines a p-dimensional manifold $\mathcal{M}$ in $D \subset X$. A *continuation process* for tracking the implicitly defined manifold $\mathcal{M}$ consists of two ingredients:

- a *predictor* providing an approximation of the local parametrization near an already computed solution $\bar{x}$,
- a *corrector* which projects a given point near $\mathcal{M}$ onto the manifold.

The most often applied continuation process is the so-called *Euler-Newton continuation*, where a (Gauss) Newton method as corrector is combined with a tangential approximation of the manifold as predictor. Not surprisingly, we shall consider an inexact Gauss Newton method as corrector.

1.1. An Inexact Gauss Newton Method. Recall that the standard Gauss Newton method for the solution of an underdetermined nonlinear problem $F(x) = 0$ is given by a sequence $x_0, x_1, \ldots$, where

$$x_{k+1} = x_k + \Delta x_k \quad \text{and} \quad \Delta x_k = -F'(x_k)^+ F(x_k) \,. \tag{3}$$

Here, $F'(x_k)^+$ denotes the pseudo inverse of the Jacobian which is well defined if $F : X \to Y$ is a Fredholm mapping of Hilbert spaces. If $F'(x_k)$ is surjective, then Δx_k is the only solution in $(\ker F'(x_k))^\perp$ of

$$F'(x_k)\Delta x_k = -F(x_k) \,. \tag{4}$$

If the nonlinear problem is very large, it is almost impossible to compute the Gauss Newton corrections Δx_k exactly. This is in particular true in the infinite dimensional

situation of a multilevel Newton (or quasilinearization) method. In an *inexact Newton method* we compute approximate solutions s_k of (4) and obtain the *inexact Newton iterates*

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

We define the *inner residuals* r_k by

$$r_k = F'(x_k)s_k + F(x_k) \ , \tag{5}$$

not to be confused with the *outer residuals* $F(x_k)$. In many applications the inexact Newton corrections s_k result from an iterative solver for the Newton equation, i.e., an *inner iteration* in contrast to Newton's method itself as the *outer iteration*. In the multilevel Newton context, we have to solve the arising infinite dimensional linear subproblems by some discretization.

Obviously, the crucial point is an appropriate matching between Newton's method (as the outer iteration) and the linear solver (the inner iteration). We want to spend as little effort as possible for the linear problems, but without loosing the quadratic convergence property of the Newton iteration. The following theorem is the basis of a relatively simple but efficient control mechanism for the inner residuals by computationally available terms. For related approaches see [DES82], [BR81], and [BR82].

To cover the underdetermined case, we restrict the corrections s_k and the directions v in the affine contravariant Lipschitz condition (6) to suitable subsets of X. Moreover, we introduce a parameter $0 \leq \beta \leq 1$ describing the "exactness" of the method. Setting $\beta = 0$ corresponds to the exact Newton iteration, while $\beta = 1$ distributes the error in equal parts on the Newton iteration and the inexact solution of the Newton equation.

THEOREM 1.1. *Let $F : D \subset X \to Y$ be a Gâteaux-differentiable mapping and $\{V(x)\}_{x\in D}$ a family of subsets $V(x) \subset X$ with $V(x) \cap \ker F'(x) = \{0\}$. We assume that there is a constant $\omega > 0$ such that*

$$\|(F'(x + tv) - F'(x))v\| \leq t\omega \, \|F'(x)v\|^2 \tag{6}$$

for all $x \in D$, $t \in [0, 1]$, and $v \in V(x)$ such that $x + v \in D$. Moreover, suppose that an inexact Newton sequence $\{x_k\}_{k=0,1,\ldots}$ exists in D such that for all $k \in \mathbb{N}$ and some $0 \leq \beta \leq 1$

a) the corrections $s_k = x_{k+1} - x_k$ are in $V(x_k)$,

b) the residuals $r_k = F(x_k) + F'(x_k)s_k$ are bounded by

$$\|r_k\| \leq \frac{\beta}{2} \min(1, \omega \, \|F'(x_k)s_k\|) \, \|F'(x_k)s_k\| \ , \tag{7}$$

c) the initial guess x_0 satisfies

$$\omega \, \|F(x_0)\| < h_{\max} := \frac{2 - \beta}{1 + \beta} \ . \tag{8}$$

Then the residuals $F(x_k)$ converge quadratically to zero.

The proof is simple and to be found in [Hoh93b]. As for almost all convergence theorems for Newton-like methods, the whole key is the judicious application of the fundamental theorem of calculus.

We call this result *affine contravariant* because it is invariant with respect to affine (contravariant) transformations of the domain X. More precisely, the constant ω in (6), which characterizes the nonlinearity of the problem, remains unchanged if we replace F by $F \circ T$, where $T : X \to X$ is an affine transformation. The whole formulation only involves the norm in the image space Y. That is why we can not prove convergence of the iterates x_k but only of the residuals $F(x_k)$.

If X is a Hilbert space and $\beta = 0$, i.e., in the exact case, we may substitute the orthogonal complement of the Jacobian's kernel for the restriction space:

$$V(x) := (\ker F'(x))^{\perp} .$$

This results in the original Gauss Newton method (3). In the inexact case $\beta \neq 0$, this choice is not realistic, since s_k is in most cases only approximately orthogonal to the $\ker F'(x_k)$. Hence, the most suitable choice for $V(x)$ appears to be the 'algorithmic' restriction

$$V(x) := \begin{cases} \{\Delta s_k\} & \text{if } x = x_k \\ \{\} & \text{if } x \notin \{x_k\}. \end{cases}$$

Introducing the *Kantorovitch quantities* $h_k := \omega \, \|F'(x_k)s_k\|$, the demand (7) on the inner residuals reads

$$\|r_k\| \leq \varepsilon_k \, \|F'(x_k)s_k\| , \quad \text{where} \quad \varepsilon_k := \frac{\beta}{2} \min(1, h_k) . \tag{9}$$

This may be easily transferred into a matching strategy for the relative residuals

$$\|r_k\| \leq \eta_k \, \|F(x_k)\|, \quad \text{where} \quad \eta_k := \frac{\varepsilon_k}{1 + \varepsilon_k} = \varepsilon_k + O(\varepsilon_k) ,$$

which obviously implies (9). This argument also holds in what follows: Up to $O(\varepsilon_k)$ the residual norm $\|F(x_k)\|$ and $\|F'(x_k)s_k\|$ are interchangeable.

Computational Estimates.. To arrive at an implementable control mechanism, we have to replace the analytic quantities h_k with computational available estimates. Following Deuflhard [Deu91], it is rather easy (see [Hoh93b]) to derive the *a posteriori* estimate

$$[h_k] := \frac{2}{1 + \beta} \frac{\|F(x_{k+1})\|}{\|F(x_k)\|} \leq h_k + O(\varepsilon_k) . \tag{10}$$

and the corresponding *a priori* estimate

$$[h_k] := \frac{1 + \beta}{2} [h_{k-1}]^2 . \tag{11}$$

Thus, we arrive at the affine contravariant matching strategy

$$[\varepsilon_k] := \frac{\beta}{2} \min(1, [h_k]) . \tag{12}$$

REMARK 1. In the algorithmic realization, the accuracy requirements should be coupled with a lower bound (depending on the prescribed accuracy for the solution of the nonlinear problem) to avoid too strict conditions for the last Newton iteration.

1.2. Stepsize Control for Continuation Methods. Next we turn to the continuation process. We define a p-dimensional predictor in the affine contravariant setting as follows.

DEFINITION 1. Let $\bar{x} \in D$ be a solution of F, $F(\bar{x}) = 0$. A *predictor* for the nonlinear problem $F(x) = 0$ at $\bar{x}$ is a mapping

$$\hat{x} : \Sigma \longrightarrow D \subset X \ ,$$

defined in an open neighbourhood Σ of $0 \in \mathbb{R}^p$ such that $\hat{x}(0) = \bar{x}$ and

$$\|F(\hat{x}(\sigma))\| \leq C\phi(\sigma) \quad \text{for all } \sigma \in \Sigma \tag{13}$$

for some constant $C > 0$ and a continuous function $\phi : \Sigma \to \mathbb{R}$ with $\phi(0) = 0$. The predictor $\hat{x}$ has *order* $q > 0$, if $\hat{x}$ fulfills condition (13) for $\phi(\sigma) = \|\sigma\|^q$, i.e.,

$$\|F(\hat{x}(\sigma))\| \leq C\|\sigma\|^q \, .$$

REMARK 2. Observe that we do not explicitly refer to the solution manifold $\mathcal{M}$ or the local parametrization x. Consequently, the dimension p of the predictor may differ from the dimension of the manifold. Thus, we can also track some submanifold of $\mathcal{M}$.

As an example, we consider the standard tangential predictor

$$\hat{x}(\sigma) = \bar{x} + \sigma t$$

for $p = 1$, where t is a normalized tangent at $\bar{x}$, i.e., $F'(\bar{x})t = 0$ and $\|t\| = 1$. For twice differentiable mappings F, it clearly has order $q = 2$, since

$$\|F(\hat{x}(\sigma))\| = \underbrace{\|F(\bar{x}) + \sigma F'(\bar{x})t\|}_{=0} + \sigma^2 O(\|F''(\bar{x})t^2\|) \, .$$

The crucial point for an effective continuation process is the coupling of the predictor and the corrector. If the stepsize σ is too big, the corrector does not converge. If σ is too small, we have to solve too many nonlinear problems. Substituting the predictor characterization (13) for the initial guess in the inexact Newton Theorem 1.1, we can derive a bound for the *maximal feasible stepsize* for which we may expect convergence of the inexact Newton iteration.

THEOREM 1.2. *Let $F : D \subset X \to Y$ be a Gâteaux differentiable mapping such that the restricted Lipschitz condition (6) holds for some $\omega > 0$ as in Theorem 1.1. In addition, let $\hat{x}$ be a predictor for $F(x) = 0$ at a solution $\bar{x} \in D$. Then the inexact Newton method initial guess $\hat{x}(\sigma)$ as defined in Theorem 1.1 converges for the for all stepsizes $\sigma \in \Sigma$ satisfying*

$$\phi(\sigma) < \frac{2-\beta}{2\omega C} h_{\max} \, , \quad \textit{where} \quad h_{\max} = \frac{2-\beta}{1+\beta} \, . \tag{14}$$

For a predictor of order q, this leads in particular to the stepsize bound

$$\|\sigma\| \leq \sqrt[q]{\frac{(2-\beta)h_{\max}}{2\omega C}} \,.$$

As for the Kantorovitch quantities h_k, we have to look for computationally available estimates for ωC to get an implementable stepsize control. Using the a posteriori estimate $[h_0]$ and inequality (14), we derive the *a posteriori estimate*

$$[\omega C] := \frac{2-\beta}{2}\frac{[h_0]}{\phi(\sigma)} \leq \frac{2-\beta}{2}\frac{h_0}{\phi(\sigma)} \leq \omega C \,.$$

For order q predictors this expression simplifies to

$$[\omega C] = \frac{2-\beta}{2}\frac{[h_0]}{\|\sigma\|^q} \,.$$

Instead of constructing an additional *a priori estimate* for ωC (probably using second order information like the curvature), we simply use the a posteriori estimate of the last continuation step. This strategy has proven to be sufficiently efficient in practice.

Thus, we arrive at a stepsize control mechanism for inexact continuation methods: Given a solution $\bar{x}$ and a predictor $\hat{x}$, look for a stepsize σ satisfying

$$\phi(\sigma) < \frac{(2-\beta)h_{\max}}{2\,[\omega C]} \tag{15}$$

and start the inexact Newton iteration with initial guess $\hat{x}(\sigma)$. If the iteration fails, contrary to expectation, use the new Kantorovitch estimate to compute $[\omega C]$ and reduce the stepsize in order to satisfy (15). For scalar parameters, $p = 1$, and an order q predictor this strategy reduces to the well known stepsize correction formula [DFK87])

$$\sigma_{\text{new}} = \sqrt[q]{\frac{h_{\max}}{[h_0]}} \cdot \sigma_{\text{old}} \,.$$

2. Object Oriented Implementation. We shall now translate the mathematical notions of the first section to corresponding classes in an object oriented implementation. While doing so we try to explain some of our design decisions.

2.1. Implementation of the Abstract Inexact Newton Method. The underlying mathematical structure is a Banach space, i.e., objects one can add, multiply with a scalar, and which are provided with a norm. Moreover, we see in the residual oriented affine contravariant approach that we only need the norm in the image space.

The first idea was to define Newton's method as a template class using the classes of the domain X and the image space Y as parameters. Besides the fact that GNU's templates had still some problems last year, the alternative approach using an abstract base class has turned out to be much more flexible. Newton's method is no "container class", but makes use of the functionality of the involved objects as elements of some Banach space (cf. [Mey92]).

The only problem is that we have to store some vectors: the right hand side $F(x_k)$, the correction s_k, and the solution x_k. Since Newton's method as an abstract class does not know the types of the domain and image space, the "user", i.e., the derived Newton method, has to provide these objects in some way.

So, what do we need to perform the iteration?

- the function evaluation $F(x)$ and the norm $\|F(x)\|$
- the evaluation of the Jacobian $F'(x)$
- the approximate solution of $F'(x)s = -F(x)$
- the update of the solution $x := x + s$

Exactly these four functions are to be found as abstract virtual methods in the base class.

```
class Newton {
public:
    Newton();

    void SetAccuracy(Real tol);
    void SetExactness(Real beta);
    void SetInitialResidual(Real f);

    Real ConvergenceFactor() const;
protected:
    virtual Bool Function(Bool first, Real& norm) = 0;
    virtual Bool Jacobian() = 0;
    virtual Bool Solver() = 0;
    virtual void AddCorrection() = 0;

    Bool Start();
    Bool Step();
    Bool Solve();

    Real RelativeAccuracy() const                { return eps; }
    Real AbsoluteAccuracy() const                { return delta; }
private:
    Real tol, beta, eps, delta;
    Real hMin, hMax, hFac, hFirst, fFirst, rhoEnd;
    Int k, kMax, red, redMax;
    Bool converged;
    RealVec f, h;

    void PrepareNextStep();
};
```

We separate the Jacobian evaluation from the solver since in some variants of Newton's method (e.g., in its affine covariant formulation) the solver is used more than once with the same Jacobian.

These functions may be realized in very different ways. So, the multilevel Newton h-p collocation for boundary value problems of ordinary differential equations (see [Hoh93b]) realizes the linear solver as an adaptive h-p collocation operating on grids, whereas the Newton method for small finite dimensional problems simply uses an LR or QR decomposition. On the other hand, the user supplied functions may employ the relative and absolute accuracy requirements *RelativeAccuracy* and *AbsoluteAccuracy* provided by the abstract method.

The implementation of the inexact Newton method looks like the following:

```
Newton::Newton() :
    kMax(30), r(0, kMax-1), h(-1, kMax+1), redMax(10),
    beta(1), tol(sqrtEpsMach), hMin(1e-3), fFirst(1), hfirst(0.1), rho(0.5)
{
    hMax = (2-beta)/(1+beta);
    hFac = 2/(1+beta);
```

```
}

Bool Newton::Start() {
    k=0;
    h.Clear();
    f.Clear();
    h(0) = hFirst;
    f(0) = fFirst;
    eps = beta * h(0) / 2;   // first required relative accuracy
    delta = eps * f(0);      // first required absolute accuracy
    Bool done = Function(true, f(0));
    if (done) {
        PrepareNextStep();
        if (converged) h(0) = hMin;
    }
    return done;
}

Bool Newton::Solve() {
    Bool done = Start();
    while (done && !converged) done = Step();
    return done;
}

Bool Newton::Step() {
    if (k>kMax) return false;
    if (!Jacobian()) return false;
    if (!Solver()) return false;
    AddCorrection();
    if (!Function(false, f(k+1)) return false;
    h(k) = hFac * f(k+1) / f(k);
    if (h(k) < hMax) return false;
    h(k+1) = sqr(h(k)) / hFac : hFirst;
    k++;
    PrepareNextStep();
    return true;
}

void Newton::PrepareNextStep() {
    converged = r(k)<=tol;           // check convergence
    f(k+1) = f(k) * h(k) / hFac;     // a priori residual
    eps = beta * h(k) / 2;           // required relative accuracy
    delta = eps * f(k+1);            // required absolute accuracy
    delta = Max(delta, rhoEnd*tol);  // soften requirement for last step
}
```

2.2. Implementation of the Continuation Process. We next sketch the classes that realize the continuation process. As claimed in the introduction, these classes should permit an easy handling of different types of solutions including bifurcations. The resulting structures are strongly influenced by experiences with rather complicated solutions occurring in the context of symmetry breaking bifurcations (code SYMCON [GH91]).

The first question is how to portray the solution of a continuation process. From our point of view, the result of a pathfollowing method is a *graph* representing the bifurcation diagram corresponding to the parameter dependent nonlinear problem. This graph consists of *points* associated with the actually computed solutions such as fixed points, bifurcations, or periodic orbits, which are connected by *edges* reflecting

the structure of the bifurcation diagram.

In the course of the continuation process, the predictor steps produce new points and edges which then have to be corrected. Hence, we distinguish two types of points and edges: *predicted* or *imaginary* points and edges which are the result of a predictor step, and *corrected* or *real* points and edges which are already corrected (or realized), e.g. by some Newton process. Accordingly, the basic class for a continuation graph looks like the following:

```
class ContGraph : public MyObject {
    friend class ContPoint;
    friend class ContEdge;
public:
    ContGraph();
    virtual ~ContGraph();

    Bool Start(ContPoint& p);
    Bool Step(Int k=1);
    Bool Continue();
    Bool Finished() const;
protected:
    List<ContEdge> edges, predictedEdges;
    List<ContPoint> points;
};
```

Next we have to ask ourselves which functionality characterizes a point of a continuation method. First, a point must have a corrector. In many cases this is some Gauss Newton process as discussed above. Moreover, a point has to provide a method which computes new edges in order to continue the pathfollowing process. In the simplest case of a regular solution point this method produces an imaginary edge corresponding to a tangential predictor. In more involved situation like symmetry breaking and Hopf bifurcations, several edges of different types are created.

```
class ContPoint : public MyObject {
public:
    virtual Bool Correct(ContGraph&) = 0;
    virtual Bool NewEdges(ContEdge&, List<ContEdge>&) = 0;
};
```

Note that the corrector may need the whole graph as argument, since it should be able to recover already computed solutions. The first argument of *NewEdges* is the edge leading to the current point. The resulting new edges are to be appended to the given list of edges.

An edge has as usual two pointers to its vertices. Moreover, there is a method *Realize* containing the predictor-corrector loop of the continuation method.

```
class ContEdge : public MyObject {
public:
    ContEdge(ContPoint& p0, ContPoint& p1);

    virtual Bool Realize(ContGraph&);
    virtual Bool Predict() = 0;
    virtual Bool CorrectStepsize() = 0;
protected:
    ContPoint *p[2];
};
```

This method *Realize* is predefined but may also be overloaded for more complicated continuation loops.

```
Bool ContEdge::Realize(Contin& contin) {
    const Int redMax = 5;
    Bool done;
    if (done = Predict()) {
        for (Int red=1, done = false; !done && red<=redMax; red++) {
            done = p[1]->Correct(contin);
            if (!done) if (!CorrectStepsize()) break;
        }
        if (done) {
            contin.points.AppendTail(p[1]);
            contin.edges.AppendTail(this);
            done = p[1]->NewEdges(this, contin.predictedEdges);
        }
    }
    return done;
}
```

2.3. Additional Remarks. Since space is limited, we have only been able to sketch the basic concepts and thus have mainly presented the abstract base classes which provide the framework for the whole package. Some questions naturally arise when deriving new continuation classes. A frequently occurring problem in C++ is connected with the static typing. How can a point (derived from *ContPoint*) get the type of a given edge (derived from *ContEdge*)? In our opinion, the best solution is the "safe downcasting" (cf. [Mey92]). As an example, consider again the abstract class *ContPoint*. So far, we have implemented three different kinds of points, namely fixed points, Hopf bifurcations, and points for periodic orbits. Thus, we can deal with the typical Hopf scenario. To handle these types of points, the base class gets three corresponding cast operators which by default return 0.

```
class OdeFixpoint;
class HopfPoint;
class ColContPoint;

class ContPoint : public MyObject {
   friend class Contin;
public:
    virtual Bool Realize(Contin&) = 0;
    virtual Bool NewEdges(ContEdge*, List<ContEdge>&) = 0;

    virtual operator OdeFixpoint* ()    { return 0; }
    virtual operator HopfPoint* ()      { return 0; }
    virtual operator ColContPoint* ()   { return 0; }
};
```

The derived class overloads the corresponding operator returning a pointer to itself.

```
class OdeFixpoint : public ContPoint {
public:
    ...

    virtual operator OdeFixpoint* () { return this; }
protected:
    ...
};
```

This mechanism may be viewed as a substitute for dynamic typing in C++. Unfortunately, it implies that the header files of the base classes have to be changed whenever a new type is introduced.

To convince the reader that the abstract framework really works in practice, we include the result of a simulation of a railway bogie. The model due to Cooperrider [Coo71] and True [Tru90] describes the motion of a bogie running with constant speed v on a perfect, stiff, level and straight track. The resulting dynamical system consists of 14 ordinary differential equations with the speed as parameter. The method employed for the solution of the periodic boundary value problems is described in [Hoh93b] and will be published elsewhere. The computed bifurcation diagram is shown in Figure 1. We encounter the typical situation of a branch of periodic solutions emanating

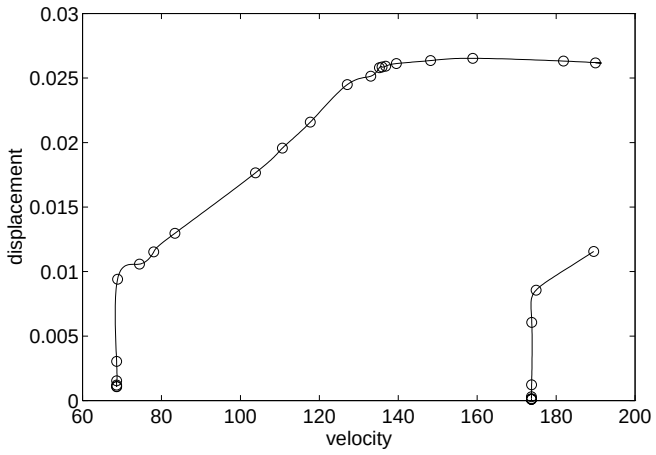


FIG. 1. *Bifurcation diagram for the bogie model*

from a trivial branch of equilibria at a Hopf bifurcation. The circles are the actually computed periodic solutions. In Figure 2 we present a three dimensional presentation

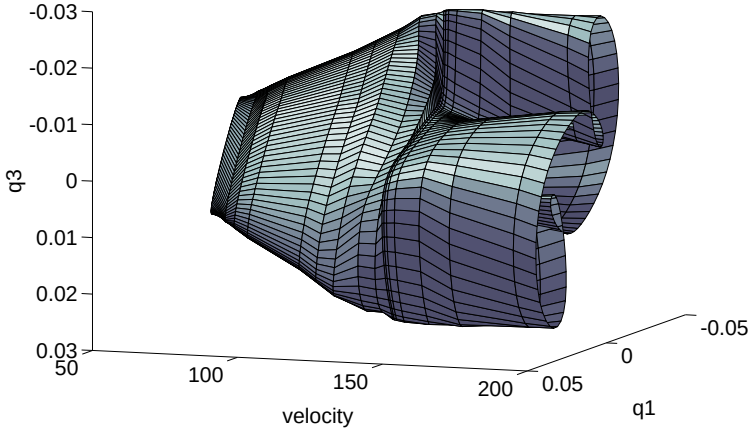


FIG. 2. *Periodic solutions of the bogie model, q_3 and q_1 versus v*

of the first branch of periodic solutions. Here, we display the components q_1 and q_3 versus the velocity v.

[BR81] R. E. Bank and D. J. Rose. Global approximate newton methods. *Numer. Math.*, 37:279–295, 1981.

[BR82] R. E. Bank and D. J. Rose. Analysis of a multilevel iterative method for nonlinear finite element equations. *Math. Comp.*, 39:453–465, 1982.

[Coo71] N. K. Cooperrider. The Hunting Behavior of Conventional Railway Trucks. *J. Eng. for Industry*, pages 1–10, 1971.

[DES82] R. S. Dembo, S. C. Eisenstat, and T. Steihaug. Inexact newton methods. *SIAM J. Numer. Anal.*, 19:400–408, 1982.

[Deu91] P. Deuflhard. Global Inexact Newton Methods for Very Large Scale Nonlinear Problems. *Impact of Computing in Science and Engineering*, 3(4):366–393, 1991.

[DFK87] P. Deuflhard, B. Fiedler, and P. Kunkel. Efficient Numerical Pathfollowing beyond Critical Points. *SIAM J. Numer. Anal.*, 18:949–987, 1987.

[FR83] J. P. Fink and W. C. Rheinboldt. On the discretization error of parametrized nonlinear equations. *SIAM J. Numer. Anal.*, 20(4):732–746, 1983.

[GH91] K. Gatermann and A. Hohmann. Symbolic Exploitation of Symmetry in Numerical Pathfollowing. *Impact of Computing in Science and Engineering*, 3(4):330–365, 1991.

[Hoh93a] A. Hohmann. An Implementation of Extrapolation Codes in C++. Technical Report TR 93–8, Konrad-Zuse-Zentrum, Berlin, 1993.

[Hoh93b] A. Hohmann. *Inexact Gauss Newton Methods for Parameter Dependent Nonlinear Problems*. PhD thesis, Freie Universität Berlin, 1993.

[Mey92] S. Meyers. *Effective C++*. Addison-Wesley Professional Computing Series, Reading, Menlo Park, New York, 1992.

[Tru90] H. True. Railway Vehicle Chaos and Asymmetric Hunting. *Proc. 12th IAVSD Symposium, Vehicle System Dynamics*, 19:625–637, 1990.