"Constantin Brâncuşi" University of Târgu Jiu ENGINEERING FACULTY
SCIENTIFIC CONFERENCE – 13 th edition – with international participation
November 07-08, 2008 – Târgu Jiu

# VISUALIZATION OF THE CONVERGENCE OF NEWTON METHOD USING MAPLE

**Mădălina Roxana Buneci**, *Assoc. Professor, University Constantin Brâncuşi*

**ABSTRACT.** Newton method is a root-finding algorithm that can often converge remarkably quickly, especially if the iteration begins "sufficiently near" the desired root. However, if the initial value is too far from a root, Newton's method can fail to converge. The purpose of this paper is to provide several Maple procedures for visualize the behavior of a given set of points used as starting points in Newton method.

## 1. A Maple procedure implementing Newton method

Newton method is an iterative method for solving general systems of m nonlinear equations:

$$f(x) = 0,$$

where $f : G \to \mathbf{R}^m$ is a continuously differentiable function on G and G is an open subset of $\mathbf{R}^m$. In the following we denote by $(x^n)_n$ a sequence of vectors in $\mathbf{R}^m$ (we reserve the subscripts to denote the component of a vector $x = (x_1, x_2,\ldots,x_m)^t \in \mathbf{R}^m$). By Jf(x) we denote the Jacobian matrix of f, this means that $Jf(x) = \left(\frac{\partial f_i}{\partial x_j}(x)\right)_{1\le i,j\le m}$

Starting with an initial guess $x^0 \in G \subset \mathbf{R}^m$, which is reasonably close to a true root $\alpha$ of the system f(x) = 0, by Newton method one finds successively better approximations of the root $\alpha$. More precisely, the methods consist in the approximation of the root $\alpha$ by $x^n$, where

$$x^n = x^{n-1} - Jf(x^{n-1})^{-1} f(x^{n-1}),\ n \ge 1 \qquad (*)$$

and where $x^0 \in G$ is an approximation "sufficiently near" $\alpha$.

If in **(*)** one left multiply with the inverse of $Jf(x^{n-1})$, then one obtains

$$Jf(x^{n-1})(x^n - x^{n-1}) = -f(x^{n-1}) \qquad (**)$$

Rather than actually computing the inverse of matrix $Jf(x^{n-1})$, one can save time by solving the preceding system for the unknown $x^n - x^{n-1}$. The sequence $(x^n)_n$ constructed above converges to $\alpha$ if the initial value $x^0$ is close enough to $\alpha$. Typically, a region which is well-behaved is located first with some other method and Newton's method is then used to improved the already known aproximation $x^0$. The algorithm can often converge remarkably quickly (quadratially for single roots, and linearly for multiple roots).

The procedure **mnewton** implement the above algorithm. The parameters of the procedure are the function f (that gives the system), the initial approximation $x^0$, the precision $\varepsilon > 0$ (iterations stop when $\|x^n - x^{n-1}\| < \varepsilon$) and the upper limit on the number of iterations Nmax.

```
> mnewton := proc(f, x0, epsilon, Nmax)
 local m,x1, x2, dx, b, fx, fx1, n, i, j, ex, r;
```

```
 m:=vectdim(x0);x1 := vector(m);x2:=vector(m);for i to m do x1[i] := x0[i] od;
 dx := vector(m); b := vector(m);  fx := jacobian(f(seq(x[i],i=1..m)), [seq(x[i],i=1..m)]);
  fx1 := matrix(m, m);
  ex := seq(x[i] = x1[i], i = 1 .. m);
  for i to m do
     for j to m do fx1[i, j] := evalf(subs(ex, fx[i, j])) od
  od;
  b := map(-evalf,f(seq(x1[i],i=1..m)));
  dx := linsolve(fx1, b, 'r');
  if r <> m then print(`The method does not apply `); RETURN(NULL) fi;
  for i to m do x2[i] := x1[i] + dx[i] od; n := 1; print(x2);
  while epsilon <= norm(dx, infinity) and n < Nmax do
     for i to m do x1[i] := x2[i] od;
     ex := seq(x[i] = x1[i], i = 1 .. m);
     for i to m do
        for j to m do fx1[i, j] := evalf(subs(ex, fx[i, j]))
        od
     od;
     b := map(-evalf, f(seq(x1[i],i=1..m))); dx := linsolve(fx1, b, 'r');
     if r <> m then print(`The method does not apply`); RETURN fi;
     for i to m do x2[i] := x1[i] + dx[i] od;
     n := n + 1;
     print(x2)
  od;
  print(`Number of iterations`, n);
  b := vector(map(evalf, f(seq(x1[i],i=1..m))));    print(`Function value`, b);
  RETURN(evalm(x2))
end;
```

## 2. Maple procedures for visualize the behavior of a given set of points used as starting points in Newton method.

If the initial approximation is too far from the root, Newton's method can fail to converge. Also the number of iterations needed for attaining a precision ε is dependent on the initial approximation.

The procedure **mnewton_vconv** returns a number in the interval [0, 1] showing how quickly the algorithm converge (0 means the algorithm does not converge) starting from a given point $x^0$. The parameters of the procedure are the function f (that gives the system), the initial approximation $x^0$, the precision $\varepsilon > 0$ (iterations stop when $||x^n - x^{n-1}|| < \varepsilon$) and the upper limit on the number of iterations Nmax.

```
> mnewton_vconv := proc(f, x0, epsilon, Nmax)
 local m,x1, x2, dx, b, fx, fx1, n, i, j, ex, r;
 m:=vectdim(x0);x1 := vector(m);x2:=vector(m);for i to m do x1[i] := x0[i] od;
 dx := vector(m); b := vector(m); fx := jacobian(f(seq(x[i],i=1..m)), [seq(x[i],i=1..m)]);
 fx1 := matrix(m, m); ex := seq(x[i] = x1[i], i = 1 .. m);
 for i to m do
     for j to m do fx1[i, j] := evalf(subs(ex, fx[i, j])) od
 od;
 b := map(-evalf,f(seq(x1[i],i=1..m)));  dx := linsolve(fx1, b, 'r');
 if r <> m then n:=Nmax+1 else
  for i to m do x2[i] := x1[i] + dx[i] od;
```

```
  n := 1;
    while epsilon <= norm(dx, infinity) and n < Nmax do
     for i to m do x1[i] := x2[i] od; ex := seq(x[i] = x1[i], i = 1 .. m);
     for i to m do
        for j to m do fx1[i, j] := evalf(subs(ex, fx[i, j]))
        od
     od;
     b := map(-evalf, f(seq(x1[i],i=1..m)));
     dx := linsolve(fx1, b, 'r');
     if r <> m then n:=Nmax+1 else
     for i to m do x2[i] := x1[i] + dx[i] od;
     n := n + 1;
   fi
  od;
 fi;
if n>=Nmax then RETURN(0) else RETURN((Nmax-n-1)/Nmax) fi
end;
```

The procedure **mnewton_vconvp** returns a number in the interval [0, 1] showing how quickly the algorithm converge to a given root (0 means the algorithm does not converge) starting from a given point $x^0$. The parameters of the procedure are the function f (that gives the system), the initial approximation $x^0$, the precision $\varepsilon > 0$ (iterations stop when $\|x^n - x^{n-1}\| < \varepsilon$), the upper limit on the number of iterations Nmax and the given root (or "good" approximation of the root) point.

```
> mnewton_vconvp := proc(f, x0, epsilon, Nmax, point)
 local m,x1, x2, dx, b, fx, fx1, n, i, j, ex, r,np,pprox,cp,q1,q2,cul;
 m:=vectdim(x0);x1 := vector(m);x2:=vector(m);for i to m do x1[i] := x0[i] od;
 dx := vector(m); b := vector(m);  fx := jacobian(f(seq(x[i],i=1..m)), [seq(x[i],i=1..m)]);
 fx1 := matrix(m, m); ex := seq(x[i] = x1[i], i = 1 .. m);
  for i to m do
     for j to m do fx1[i, j] := evalf(subs(ex, fx[i, j])) od
  od;
  b := map(-evalf,f(seq(x1[i],i=1..m))); dx := linsolve(fx1, b, 'r');
  if r <> m then n:=Nmax+1 else    for i to m do x2[i] := x1[i] + dx[i] od; n := 1;
  while epsilon <= norm(dx, infinity) and n < Nmax do
   for i to m do x1[i] := x2[i] od; ex := seq(x[i] = x1[i], i = 1 .. m);
     for i to m do
        for j to m do fx1[i, j] := evalf(subs(ex, fx[i, j]))
        od
     od;
     b := map(-evalf, f(seq(x1[i],i=1..m))); dx := linsolve(fx1, b, 'r');
     if r <> m then n:=Nmax+1 else
     for i to m do x2[i] := x1[i] + dx[i] od; n := n + 1;
   fi
  od;
 fi;
 if n>=Nmax then RETURN(0) else for j from 1 to m do dx[j]:=point[j]-x2[j] od;
   if norm(dx,infinity)>epsilon/2 then RETURN(0) else RETURN((Nmax-n+1)/Nmax) fi;
 fi;
end;
```

The procedure **mnewton_points** returns a list of roots to which the Newton methods converges starting from a set of initial approximations. The parameters of the procedure are the function f (that gives the system), the list of initial approximations a, the precision $\varepsilon > 0$ (iterations stop when $||x^n - x^{n-1}|| < \varepsilon$), the upper limit on the number of iterations Nmax.

```
> mnewton_points := proc(f, a, epsilon, Nmax)
  local m,x1, x2, dx, b, fx, fx1, n, i, j, jv,ex, r,ip,p,test;
  p:=[];m:=vectdim(a[1]);x1 := vector(m);x2:=vector(m);
  for ip to nops(a) do for i to m do x1[i] := a[ip][i] od;
  dx := vector(m);b := vector(m); fx := jacobian(f(seq(x[i],i=1..m)), [seq(x[i],i=1..m)]);
  fx1 := matrix(m, m);ex := seq(x[i] = x1[i], i = 1 .. m);
  for i to m do
      for j to m do fx1[i, j] := evalf(subs(ex, fx[i, j])) od
  od;
  b := map(-evalf,f(seq(x1[i],i=1..m)));   dx := linsolve(fx1, b, 'r');
  if r <> m then n:=Nmax+1 else;
    for i to m do x2[i] := x1[i] + dx[i] od; n := 1;
    while epsilon <= norm(dx, infinity) and n < Nmax do
       for i to m do x1[i] := x2[i] od; ex := seq(x[i] = x1[i], i = 1 .. m);
       for i to m do
          for j to m do fx1[i, j] := evalf(subs(ex, fx[i, j]))
          od
       od;
       b := map(-evalf, f(seq(x1[i],i=1..m)));dx := linsolve(fx1, b, 'r');
       if r <> m then n:=Nmax+1 else
       for i to m do x2[i] := x1[i] + dx[i] od; n := n + 1;
       fi;
     od;
   fi;
   if n < Nmax then test:=1;i:=1;
   while i <= nops(p) do for j from 1 to m do dx[j]:=p[i][j]-x2[j] od;
      if norm(dx,infinity)^2<epsilon then test:=0; i:=nops(p)+1 fi; i:=i+1 od;
       if test=1 then p:=[op(p),evalm(x2)] fi
      fi
   od;
 RETURN(p)
end;
```

The procedure **mnewton_colors** returns a list of three numbers in [0, 1] that can be used to associate a color with the RGB models. showing how quickly the algorithm converge to a given root (0 means the algorithm does not converge) starting from a given point $x^0$. The parameters of the procedure are the function f (that gives the system), the initial approximation $x^0$, the precision $\varepsilon > 0$ (iterations stop when $||x^n - x^{n-1}|| < \varepsilon$), the upper limit on the number of iterations Nmax and the given list of roots (or "good" approximation of the roots) p. Each roots is associated with a color (according to its position in the input list p). The procedure return a triple needed to define a RGB color. The return color correspond to the root to which the algorithm converge starting from the initial approximation $x^0$. A darker color indicates a greater number of iterations to get close to the corresponding root.

```
> mnewton_colors := proc(f, x0, epsilon, Nmax,p)
  local m,x1, x2, dx, b, fx, fx1, n, i, j, ex, r,np,pprox,cp,q1,q2,cul;
```

```
  m:=vectdim(x0);x1 := vector(m);x2:=vector(m);for i to m do x1[i] := x0[i] od;
  dx := vector(m); b := vector(m); fx := jacobian(f(seq(x[i],i=1..m)), [seq(x[i],i=1..m)]);
  fx1 := matrix(m, m); ex := seq(x[i] = x1[i], i = 1 .. m);
   for i to m do
      for j to m do fx1[i, j] := evalf(subs(ex, fx[i, j])) od
   od;
   b := map(-evalf,f(seq(x1[i],i=1..m))); dx := linsolve(fx1, b, 'r');
   if r <> m then n:=Nmax+1 else    for i to m do x2[i] := x1[i] + dx[i] od; n := 1;
  while epsilon <= norm(dx, infinity) and n < Nmax do
      for i to m do x1[i] := x2[i] od; ex := seq(x[i] = x1[i], i = 1 .. m);
      for i to m do
         for j to m do fx1[i, j] := evalf(subs(ex, fx[i, j]))
         od
      od;
      b := map(-evalf, f(seq(x1[i],i=1..m))); dx := linsolve(fx1, b, 'r');
      if r <> m then n:=Nmax+1 else
      for i to m do x2[i] := x1[i] + dx[i] od;
      n := n + 1;
      fi
    od;
   fi;
   if n>=Nmax then cp:=0 else for j from 1 to m do dx[j]:=p[1][j]-x2[j] od;
     pprox:=norm(dx,infinity);cp:=1;
     for i from 2 to nops(p) do
       for j from 1 to m do dx[j]:=p[i][j]-x2[j]od;
       if norm(dx,infinity)<pprox then pprox:=norm(dx,infinity); cp:=i fi
     od;
    fi;
    np:=floor(nops(p)^(1/3))+1;
    cul:=[0,0,0];
    if np>1 then cul[1]:=irem(cp,np,'q1')/(np-1)*(Nmax-n+1)/Nmax;
    cul[2]:=irem(q1,np,'q2')/(np-1)*(Nmax-n+1)/Nmax;
    cul[3]:=irem(q2,np)/(np-1)*(Nmax-n+1)/Nmax;
  fi;
  RETURN(cul)
end;
```

The procedure **det_list** divides the rectangle [xmin,xmax]×[ymin, ymax] into nx × ny subrectangles and return the list of the vertices of these subrectangles.

```
> det_list:=proc(xmin,xmax,ymin,ymax,nx,ny)
  local a, i,j;
  a:=[];
  for i from 0 to nx do for j from 0 to ny do
    a:=[op(a),vector([xmin+i*(xmax-xmin)/nx,ymin+j*(ymax-ymin)/ny])]
  od od;
  RETURN(a)
end;
```

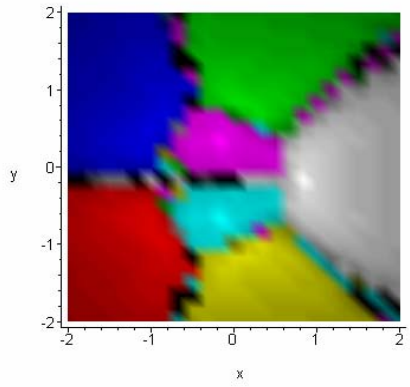
**Example**

```
> with(linalg):
```

```
> with(plots):
> f:=(x,y)->[x^7-x^5*y^2-5*y^4*x^3-x^2-3*y^6*x+y^2+1/4*x^5-
5/2*x^3*y^2+5/4*y^4*x-1/4,-5*x^4*y^3+y^7-y^5*x^2+5/2*x^2*y^3-3*x^6*y-2*x*y-
5/4*x^4*y-1/4*y^5];
```

$$f := (x, y) \to \left[ x^7 - x^5 y^2 - 5 y^4 x^3 - x^2 - 3 y^6 x + y^2 + \frac{1}{4} x^5 - \frac{5}{2} x^3 y^2 + \frac{5}{4} y^4 x - \frac{1}{4}, \right.$$
$$\left. -5 x^4 y^3 + y^7 - y^5 x^2 + \frac{5}{2} x^2 y^3 - 3 x^6 y - 2 y x - \frac{5}{4} x^4 y - \frac{1}{4} y^5 \right]$$

```
> a:=det_list(-2,2,-2,2,25,25):
> p:= mnewton_points(f,a,10^(-8),20):
> plot3d(0,-2 .. 2, -2 .. 2, grid=[25, 25], style=patchnogrid,axes=framed,color=[proc(x,y)
mnewton_colors(f,vector([x,y]),10^(-5),20,p)[1] end proc, proc(x,y)
mnewton_colors(f,vector([x,y]),10^(-5),20,p)[2] end proc, proc(x,y)
mnewton_colors(f,vector([x,y]),10^(-5),20,p)[3] end proc], labels=[x,y,z],
scaling=CONSTRAINED, view=0..10^(-5));
```




**References**


[1] P. Deuflhard, *Newton Methods for Nonlinear Problems*. Affine Invariance and Adaptive Algorithms. Springer Series in Computational Mathematics, Vol. **35**. Springer, Berlin, 2004.

[2] M.B. Monagan, K.O. Geddes, K.M. Heal, G. Labahn, S.M. Vorkoetter, J. McCarron and P. DeMarco, *Maple 8 Advanced Programming Guide*. Waterloo Maple Inc., Waterloo, Ontario, Canada, 2002.