OpenCASCADE Root-Finding Algorithm

eryar@163.com

Abstract. A root-finding algorithm is a numerical method, or algorithm, for finding a value x such that f(x)=0, for a given function f. Such an x is called a root of the function f. In OpenCASCADE math package, implemente Newton-Raphson method to find the root for a function. The algorithm can be used for finding the extrema value for curve or surface, .i.e Point Inversion, find the parameter for a point on the curve or surface. The paper focus on the usage of OpenCASCADE method and its application.

Key Words. OpenCASCADE, Extrema, Newton-Raphson, Root-Finding, FunctionRoot

1. Introduction

代数方程求根问题是一个古老的数学问题，早在 16 世纪就找到了三次、四次方程的求根公式。但直到 19 世纪才证明 n>=5 次的一般代数方程式不能用代数公式求解。在工程和科学技术中，许多问题常常归结为求解非线性方程的问题，因此，需要研究用数值方法求得满足一定精度的代数方程式的近似解。

我国古代宋朝数学家秦九韶在他 1247 年所著的《数书九章》中，给出一个求代数方程根的近似方法，这个方法一般书上都称为和纳 Horner 方法（英国数学家 W.G.Horner）。实际上 Horner 在 1819 年才提出这个方法，比秦九韶晚五百多年。每当看到教科书中这样的介绍不知是该骄傲，还是该嗤之以鼻。古人发明创造的东西比外国人早，而现在国内用于 CAD、CAM 的软件大都是国外进口的，像 CATIA，AutoCAD，Pro/E，UG NX，SolidWorks，AVEVA Plant/Marine，Intergraph，ACIS，Parasolid……等等不胜枚举，很少看到中国软件的身影。而这些软件广泛应用于航空、造船、机械设计制造、工厂设计等各个行业，每年的软件授权费用不知几何？衷心希望当代国人奋发作为，为世界增添色彩。

闲话少说，本文主要关注非线性方程的数值解法，重点介绍了 Newton-Rophson 法及在 OpenCASCADE 中应用，即求点到曲线曲面的极值，也就是曲线曲面点的反求参数问题。对数值算法感兴趣的读者，可以参考《数值分析》、《计算方法》之类的书籍以获取更详细信息。

2. Numerical Methods

方程求根的方法有很多，在《数学手册》中列举了如下一些方法：

- 秦九韶法；
- 二分法；
- 迭代法；
- 牛顿法 Newton's Method；
- 弦截法；
- 抛物线法；
- 林士谔－赵访熊法；

其中二分法是求实根的近似计算中行之有效的最简单的方法，它只要求函数是连续的，因此它的使用范围很广，并便于在计算机上实现，但是它不能求重根也不能求虚根，且收敛较慢。

Newton 法在单根邻近收敛快，具有二阶收敛速度，但 Newton 法对初值要求比较苛刻，即要求初值选取充分靠近方程的根，否则 Newton 法可能不收敛。扩大初值的选取范围，可采用 Newton 下山法。

Newton's Method 的实现原理的演示动画如下图所示：

http://upload.wikimedia.org/wikipedia/commons/e/e0/NewtonIteration_Ani.gif

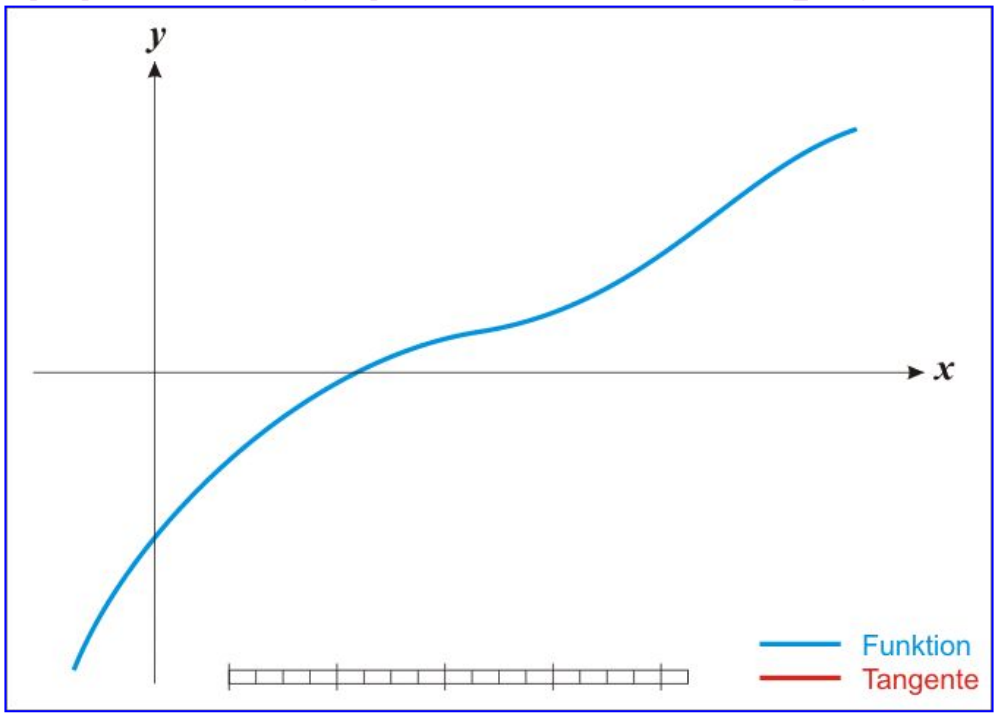


Figure 2.1 Newton's Method(Newton-Raphson)

由上面的动画可以清楚理解 Newton 法的原理。用数学的文字描述如下：设 f(x)二次连续可导，xk 是 f(x)=0 的第 k 次近似解。我们用曲线 y=f(x)过点(xk,yk)的切线 Lk：

$$y = f(x_k) + f'(x_k)(x - x_k)$$

来近似曲线 f(x)。取 Lk 与 X 轴的交点为 f(x)=0 的第 k+1 次近似解为：

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

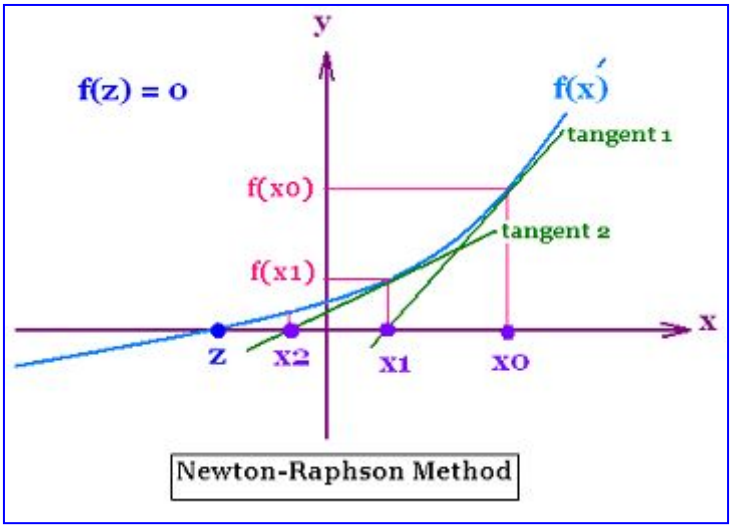


Figure 3.2 Newton-Raphson Method

其中：

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

称为 Newton 公式。

Newton 法对初值 x0 要求苛刻，在实际应用中往往难以满足。Newton 下山法是一种降低对初值要求的修正 Newton 法。

关于 Newton 方法的公开课的视频我找到网易上有节课，介绍了 Newton 方法的原理及用法，网址为：http://v.163.com/movie/2006/8/T/V/M6GLI5A07_M6GLLGSTV.html，在后半部分。老师用实际例子来讲解还挺有意思的，感兴趣的读者也可以完整地看看，也可复习下微积分的知识点。

3. OpenCASCADE Function Root

OpenCASCADE 的 math 包中实现了方程求根的算法，相关的类有 math_FunctionRoot，math_FunctionRoots，math_NewtonFunctionRoot 等。在《Fundation Classes User's Guide》中有对通用数学算法的介绍，即 OpenCASCADE 中实现了常见的数学算法：

- 求解线性代数方程的根的算法；
- 查找方程极小值的算法；
- 查找非线性方程（组）的根；
- 查找对角矩阵的特征值及特征向量的算法；

所有的数学算法以相同的方式来实现，即：在构造函数中来做大部分的计算，从而给出适当的参数。所有相关数据都保存到结果对象中，因此所有的计算尽量以最高效的方式来求解。函数 IsDone()表明计算成功。如下所示分别为采用不同的算法来计算如下方程在[0，2]区间上的根：

$$f(x) = x^6 - x - 1$$
$$f'(x) = 6x^5 - 1$$

实现程序代码如下所示：

```
/*
*    Copyright (c) 2014 eryar All Rights Reserved.
*
*       File    : Main.cpp
*       Author  : eryar@163.com
*       Date    : 2014-10-20 18:52
*       Version : 1.0v
*
*    Description : Test OpenCASCADE function root algorithm.
*
*     Key words : OpenCASCADE, Newton-Raphson, Root-Finding Algorithm,
FunctionRoot
*/

#define WNT

#include <Precision.hxx>

#include <math_FunctionWithDerivative.hxx>

#include <math_BissecNewton.hxx>
#include <math_FunctionRoot.hxx>
#include <math_NewtonFunctionRoot.hxx>

#pragma comment(lib, "TKernel.lib")
#pragma comment(lib, "TKMath.lib")

class TestFunction : public math_FunctionWithDerivative
{
public:
    virtual    Standard_Boolean    Value(const    Standard_Real    X,
Standard_Real& F)
    {
```

```
        F = pow(X, 6) - X - 1;

        return Standard_True;
    }

    virtual Standard_Boolean Derivative(const Standard_Real X,
Standard_Real& D)
    {
        D = 6 * pow(X, 5) - 1;

        return Standard_True;
    }

    virtual Standard_Boolean Values(const Standard_Real X,
Standard_Real& F, Standard_Real& D)
    {
        Value(X, F);

        Derivative(X, D);

        return Standard_True;
    }
};

void TestFunctionRoot(void)
{
    TestFunction aFunction;

    math_FunctionRoot aSolver1(aFunction, 1.5, 0.0, 0.0, 2.0);

    math_BissecNewton aSolver2(aFunction, 0.0, 2.0, 0.0);

    math_NewtonFunctionRoot aSolver3(aFunction, 1.5,
Precision::Confusion(), Precision::Confusion());

    std::cout << aSolver1 << std::endl;
    std::cout << aSolver2 << std::endl;
    std::cout << aSolver3 << std::endl;
}

int main(int argc, char* argv[])
{
    TestFunctionRoot();

    return 0;
}
```

由上述代码可知，要想使用求根算法，必须从 math_FunctionWithDerivative 派生且重载其三个纯虚函数 Value(), Derivative(), Values()，在这三个纯虚函数中计算相关的值及导数值即可。所以实际使用时，正确重载这三个函数是正确使用求根算法的关键。

求根用了三个不同的类，即三种方法来实现：

- math_FunctionRoot：即 Newton-Raphson 法；
- math_BissecNewton：是 Newton-Raphson 和二分法的组合算法；
- math_NewtonFunctionRoot：Newton Method；

计算结果如下图所示：

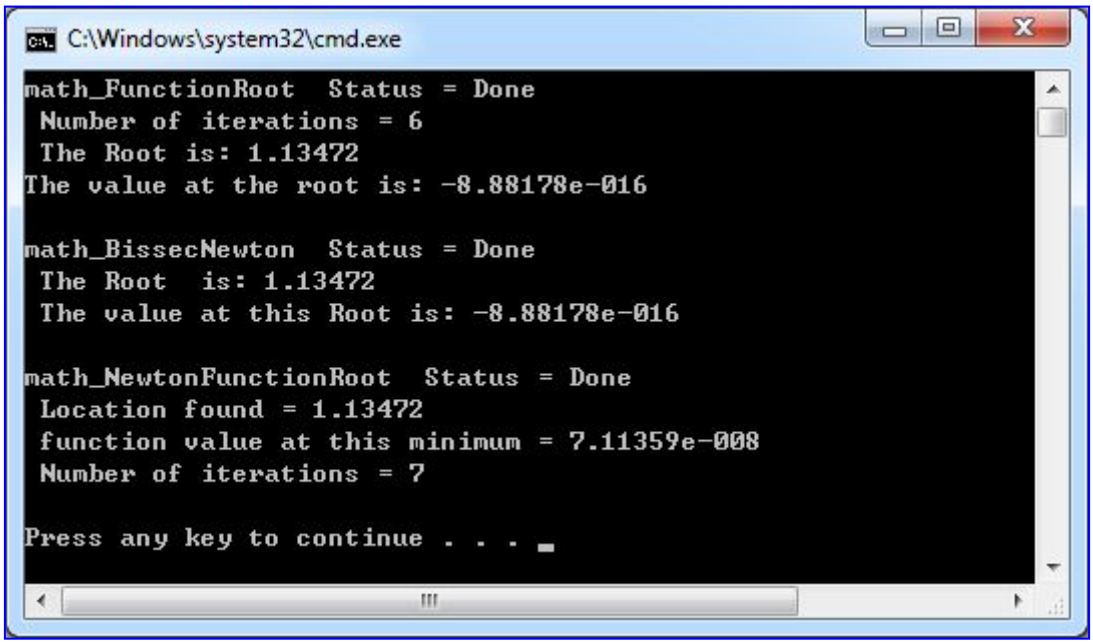


Figure 3.1 Root-Finding result of OpenCASCADE

由计算结果可知，三种方法计算的结果相同，都是 1.13472，与书中结果吻合。但是 math_NewtonFunctionRoot 的迭代次比 math_FunctionRoot 多一次，且计算精度要低很多。

使用 math_BissecNewton 求根不用设置初始值，比较方便，精度与 math_FunctionRoot 一致。

4. Application

在工程和科学技术中，许多问题常常归结为求解非线性方程的问题。在 OpenCASCADE 中的应用更多了，从下面一张类图可见一斑：

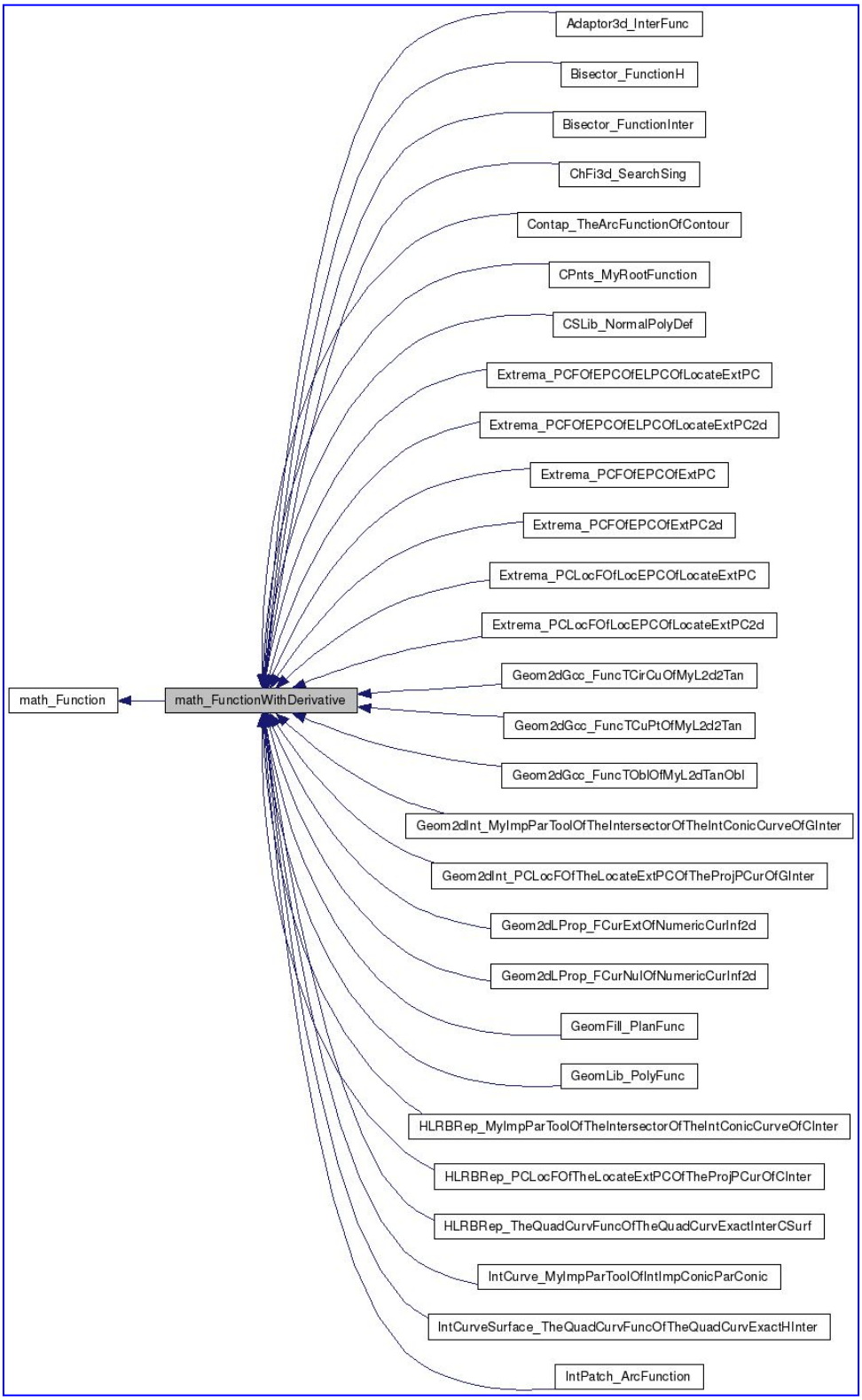
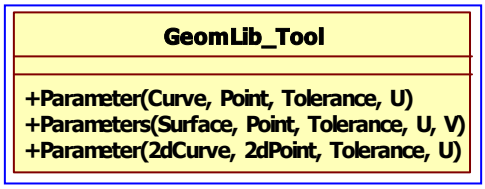


Figure 4.1 math_FunctionWithDerivative class diagram

由图可知，从类 math_FunctionWithDerivative 派生出了很多可导函数的类，这些函数都可用于求根的类中，从而计算出对应方程的根。

下面给出一个实际应用，即曲线曲面上点的反求问题，来说明如何应用上述求根算法来解决实际的问题。由于曲线曲面的参数表示法，通过参数 u 或(u,v)可以方便地求出曲线上的点或曲面上的点。若给定一个点 P(x,y,z)，假设它在 p 次 NURBS 曲线 C(u)上，求对应的参数 u'使得 C(u')=P，这个问题称为点的反求（point inverse）。在 OpenCASCADE 中提供了简单的函数接口来实现点的反求，使用类为 GeomLib_Tool：



如何将点的反求问题归结为方程求根的问题，就要根据条件来建立方程了。一种简单并完全可以解决这一问题的方法是：利用 Newton 迭代法来最小化点 P 和 C(u)的距离，如下图所示。如果最小距离小于一个事先指定的精度值，则认为点 P 在曲线上。这种方法有效地解决了更一般的“点在曲线上的投影”的问题。

因为方程求根的 Newton 方法需要指定初值 u0，所以可按如下方法得到一个用于 Newton 法的初值 u0：

- 如果已知点 P 在给定精度内位于曲线上，则用强凸包性确定候选的段，对于一般的点到曲线的投影问题，则选择所要的段作为候选段；
- 在每个候选段上，计算 n 个按参数等间隔分布的点。计算出所有这些点和点 P 的距离，选择其中距点 P 最近的点的参数作为 u0。点数 n 一般利用某种启发的方法来选择。

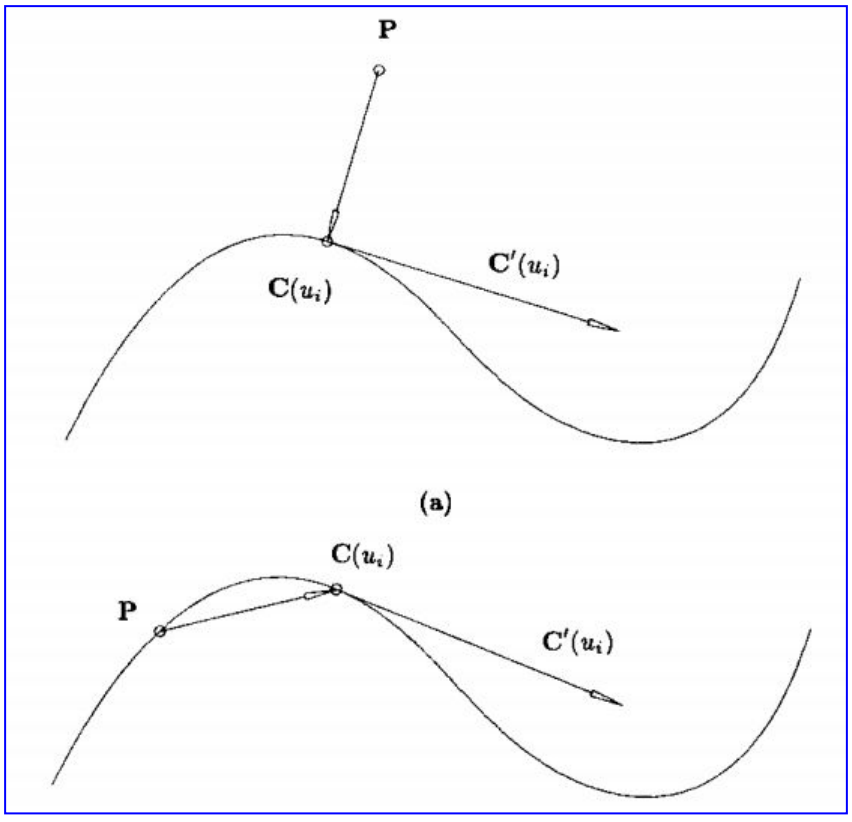


Figure 4.2 Point projection and Point inversion

需要强调的是使用 Newton 方法，一个好的初值对于迭代的收敛性及收敛速度是非常重要的。现在假设已经确定了初值 u0，利用数量积定义函数：

$$f(u)=C'(u)\cdot(C(u)-P)$$

不管 P 点是否位于曲线上，当 f(u)=0 时，点 P 到 C(u)的距离达到最小。对 f(u)求导得：

$$f'(u)=C''(u)\cdot(C(u)-P)+\left|C'(u)\right|^2$$

代入 Newton 迭代公式得：

$$u_{i+1}=u_i-\frac{f(u_i)}{f'(u_i)}=u_i-\frac{C'(u)\cdot(C(u)-P)}{C''(u)\cdot(C(u)-P)+\left|C'(u)\right|^2}$$

在 OpenCASCADE 中曲线点的反求主要是使用了派生自 math_FunctionWithDerivative 的类 Extrema_PCFOfEPCOfExtPC，类图如下所示：

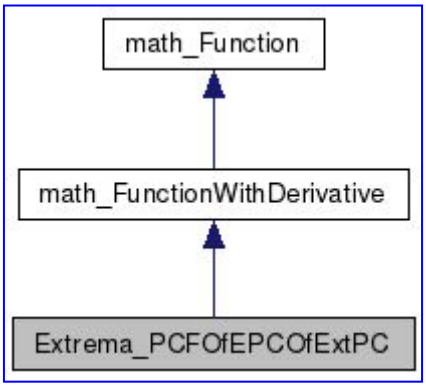


Figure 4.3 class diagram for point inverstion

所以需要实现三个纯虚函数 Value(), Derivative(), Values()，将其实现代码列出如下所示：

```
Standard_Boolean Extrema_FuncExtPC::Value (const Standard_Real U, Standard_Real&
F)
{
  if (!myPinit || !myCinit) Standard_TypeMismatch::Raise();
  myU = U;
  Vec D1c;
  Tool::D1(*((Curve*)myC),myU,myPc,D1c);
  Standard_Real Ndu = D1c.Magnitude();
  if (Ndu <= Tol) { // Cas Singulier (PMN 22/04/1998)
    Pnt P1, P2;
    P2 = Tool::Value(*((Curve*)myC),myU + delta);
    P1 = Tool::Value(*((Curve*)myC),myU - delta);
    Vec V(P1,P2);
    D1c = V;
    Ndu = D1c.Magnitude();
    if (Ndu <= Tol) {
```

```
      Vec aD2;
      Tool::D2(*((Curve*)myC),myU,myPc,D1c,aD2);
      Ndu = aD2.Magnitude();

      if(Ndu  <= Tol)
       return Standard_False;
      D1c = aD2;
    }
  }

  Vec PPc (myP,myPc);
  F = PPc.Dot(D1c)/Ndu;
  return Standard_True;
}
//=============================================================================

Standard_Boolean   Extrema_FuncExtPC::Derivative   (const   Standard_Real   U,
Standard_Real& D1f)
{
  if (!myPinit || !myCinit) Standard_TypeMismatch::Raise();
  Standard_Real F;
  return Values(U,F,D1f);  /* on fait appel a Values pour simplifier la
                  sauvegarde de l'etat. */
}
//=============================================================================

Standard_Boolean Extrema_FuncExtPC::Values (const Standard_Real U, Standard_Real&
F, Standard_Real& D1f)
{
  if (!myPinit || !myCinit) Standard_TypeMismatch::Raise();
  myU = U;
  Vec D1c,D2c;
  Tool::D2(*((Curve*)myC),myU,myPc,D1c,D2c);

  Standard_Real Ndu = D1c.Magnitude();
  if (Ndu <= Tol) {// Cas Singulier (PMN 22/04/1998)
    Pnt P1, P2;
    Vec V1;
    Tool::D1(*((Curve*)myC),myU+delta, P2, V1);
    Tool::D1(*((Curve*)myC),myU-delta, P1, D2c);
    Vec V(P1,P2);
    D1c = V;
    D2c -= V1;
    Ndu = D1c.Magnitude();
    if (Ndu <= Tol) {
      myD1Init = Standard_False;
      return Standard_False;
    }
  }

  Vec PPc (myP,myPc);
```

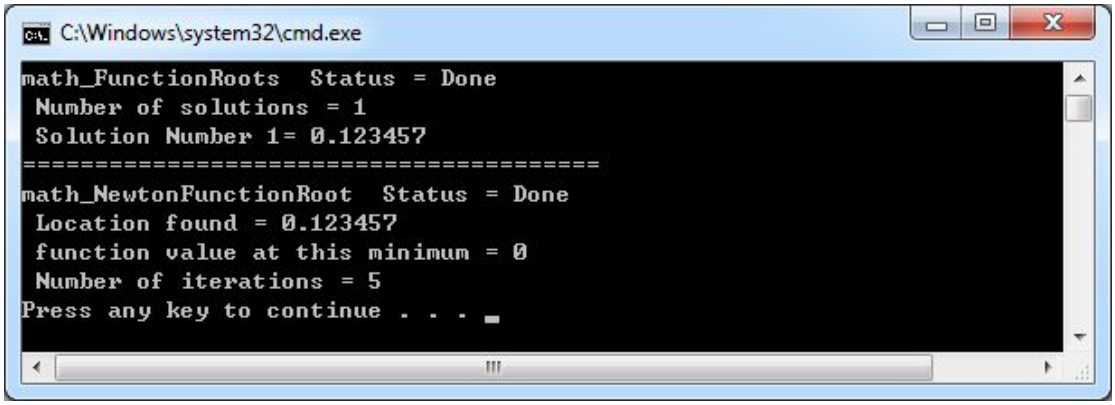
```
  F = PPc.Dot(D1c)/Ndu;
  D1f = Ndu + (PPc.Dot(D2c)/Ndu) - F*(D1c.Dot(D2c))/(Ndu*Ndu);

  myD1f = D1f;
  myD1Init = Standard_True;
  return Standard_True;
}
```

根据代码可知，实现原理与上述一致。下面给出一个简单的例子，来说明及方便调试点的反求算法。示例程序代码如下所示：

```
/*
*    Copyright (c) 2014 eryar All Rights Reserved.
*
*        File    : Main.cpp
*        Author  : eryar@163.com
*        Date    : 2014-10-20 18:52
*        Version : 1.0v
*
*    Description : Test OpenCASCADE function root algorithm.
*
*      Key words : OpenCascade, Extrema, Newton's Method
*/

#define WNT

#include <math_FunctionRoots.hxx>
#include <math_NewtonFunctionRoot.hxx>

#include <Extrema_PCFOfEPCOfExtPC.hxx>

#include <GC_MakeCircle.hxx>

#include <GeomAdaptor_Curve.hxx>

#pragma comment(lib, "TKernel.lib")
#pragma comment(lib, "TKMath.lib")
#pragma comment(lib, "TKG3d.lib")
#pragma comment(lib, "TKGeomBase.lib")


void TestExtrema(void)
{
    Handle_Geom_Curve aCircle = GC_MakeCircle(gp::XOY(), 2.0);

    GeomAdaptor_Curve aCurve(aCircle);

    Extrema_PCFOfEPCOfExtPC aFunction(aCircle->Value(0.123456789), aCurve);

    math_FunctionRoots aSolver1(aFunction, -2.0, 2.0, 10);
    math_NewtonFunctionRoot aSolver2(aFunction, 0.0, 0.0, 0.0);
```

```
    aSolver1.Dump(std::cout);
    std::cout << "=====================================" << std::endl;
    aSolver2.Dump(std::cout);
}

int main(int argc, char* argv[])
{
    TestExtrema();

    return 0;
}
```

根据圆上一点，求出对应的参数值，计算结果如下所示：



5. Conclusion

Newton 法可以选作对导数能有效求值，且导数在根的邻域中连续的任何函数方程的求根方法。Newton 法在单根邻近收敛快，精度高，具有二阶收敛速度，但 Newton 法对初值要求比较高，即要求初值选取充分靠近方程的根，否则 Newton 法可能不收敛。

OpenCASCADE 的 math 包中提供了求根的几种实现算法，虽然代码有些乱，但是这种抽象的思想还是相当不错的，便于扩展应用。理解了 math_FunctionRoot 的算法，进而可以理解从 math_FunctionWithDerivative 派生的类的原理了。

通过曲线上点的反求问题引出使用求根算法的具体实例，从中可以看出关键还是要将实际问题抽象成一个方程。有了方程，根据 Newton 迭代公式，求出相应的值和导数值，就可以得到方程的高精度的根了。

对数值算法感兴趣的读者，可以参考《计算方法》、《数值分析》等相关书籍，从而可以在理解 OpenCASCADE 的代码的基础上，可以自己来实现相关算法。

6. References

1. 数学手册编写组. 数学手册. 高等教育出版社. 1979
2. 赵罡，穆国旺，王拉柱译. 非均匀有理 B 样条. 清华大学出版社. 2010
3. Les Piegl, Wayne Tiller. The NURBS Book. Springer-Verlag. 1997
4. 易大义，沈云宝，李有法编. 计算方法. 浙江大学出版社. 2002
5. 易大义，陈道琦编. 数值分析引论. 浙江大学出版社. 1998
6. 李庆杨，王能超，易大义.数值分析.华中理工大学出版社. 1986
7. 同济大学数学教研室. 高等数学(第四版). 高等教育出版社. 1996
8. Newton's Method video:
   http://v.163.com/movie/2006/8/T/V/M6GLI5A07_M6GLLGSTV.html
9. http://en.wikipedia.org/wiki/Root-finding_algorithm
10. http://mathworld.wolfram.com/Root-FindingAlgorithm.html
11. http://mathworld.wolfram.com/NewtonsMethod.html