Applications of Symbolic computation in C++ Programming Language

Satabaldiyev Askar Bakytzhanuly
SuleymanDemirel University
Almaty, Kazakhstan

Latuta Konstantin Nikolayevich
SuleymanDemirel University
Almaty, Kazakhstan

ABSTRACT
This paper contains the brief information about symbolic computation techniques. The location of symbolic computation within Computer Language classification is defined. Programming languages (PL) can be divided into two main areas: imperative PL and declarative PL. Declarative PL is generally used for logical and functional programming (ex: PROLOG, LISP, ML, Haskell,…) Symbolic computation techniques are commonly derived from both, logical and functional programming.
Imperative programming languages are practically used for numerical computations. Some of well-known examples of imperative PL's are C, C++, Java, MATLAB, etc… But for last few years, the programming languages mentioned above have been oriented to solve problems symbolically.
The general idea of this paper is to provide the implementation of well-known mathematical problems symbolically solved on the basis of C++ programming language. It follows some typical examples to illustrate the properties of symbolic C++.

General Terms
Programming languages, symbolic computation, logical programming, functional programming

Keywords
Symbolic computation, C++, "symbolic++.h"

1. INTRODUCTION
Symbolic computation relates to algorithms and software for manipulating mathematical expressions and equations in symbolic form, as opposed to numeric computation that deals with
approximations of specific numerical quantities expressed by those symbols. The software that implements symbolic calculations might be user for symbolic differentiation, substitution one expression into another, finding polynomial, etc...generally for every computation with mathematical terms for well-known algorithms. Symbolic computations are more exact rather than numeric solutions. The reason is that numeric solutions are more approximate and they are oriented for specific cases, but symbolic computations are more general. Because of this, symbolic computation methods may need to take long time and resources to solve problems. Numeric computation methods are much faster. Some problems can be better solved using symbolical way, while other can be solved better in a numerical way[11].
In Computer Science there exist many programming languages, which are used for numerical applications as solution to problems.
Programming languages (PL) can be divided into two main areas: imperative PL and declarative PL. Declarative PL is generally used for logical and functional programming (ex: PROLOG, LISP, ML, Haskell,…) Symbolic computation techniques are commonly derived from both, logical and functional programming.
Imperative programming languages are practically used for numerical computations. Some of well-known examples of imperative PL are C, C++, Java, MATLAB, etc… But for last few years, the programming languages mentioned above have been oriented to solve problems symbolically.
General classification of programming languages
is illustrated in Figure 1

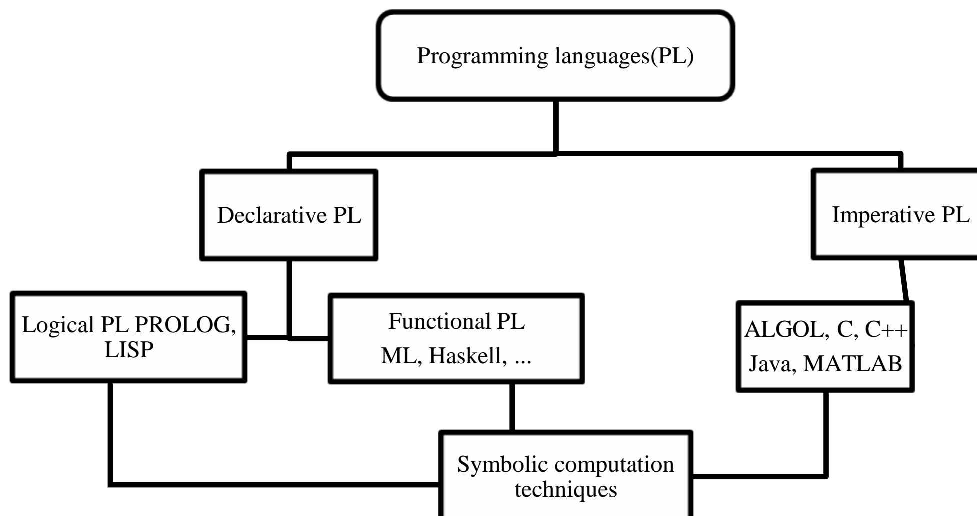

Figure1: Classification of programming languages

Symbolic computation techniques (SCT) are created from knowledge logical and functional programming techniques. Some examples of SCT open-source, cost free tools are given in the following list: Axiom, Bergmann, Calc, CoCoA, DoCon, DCAS, Eigenmath, Franklin Math, FriCAS, GAP, etc…

Also there are more than 30 commercial SCT tools. Detailed information can be found in reference [1], [2], [3], [4], [5], [6].

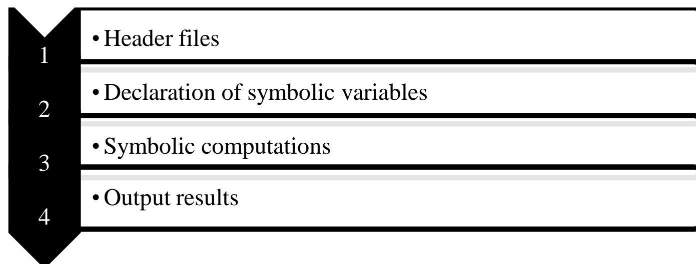

Figure 2: General procedure for symbolic C++ program

2. APPLICATION OF SYMBOLIC C++

This chapter demonstrates the application of symbolic C++ programming for the most common mathematical subjects, such as Vector Algebra, Algebra, Polynomials, and Series.

2.1 Vector Algebra

Table 1.List of functions, descriptions and source codes for examples from vector algebra

Function	Description	Source code
+ -	Vector addition and subtraction	veca.cpp
|	Scalar product	sp.cpp
%	Cross product	cp.cpp
	Gradient	grad.cpp
	Divergence	diverge.cpp

2.2 Algebra

Table 2.List of functions, descriptions and source codes for examples from algebra

Function	Description	Source code
Df	Differentiation	diff.cpp
Integrate	Integration	integ.cpp

2.3 Polynomial

Table 3.List of functions, descriptions and source codes for examples with polynomials

Function	Description	Source code
Diff	Multivariable polynomial differentiation	poly.cpp
Legendre	Solution of Legendre differential equations	legendre.cpp

There are not much publications about symbolic C++. There is only one textbook and some papers [7],[8],[9]

There is an extensive library for symbolic C++, namely “GiNaC”[10], which is widely used for projects with symbolic C++. The following examples illustrate the basic idea of symbolic computation applying the core library “symbolicc++.h”, without using extensive “GiNaC” library.

2.4 Series

Table 4. List of functions, descriptions and source codes for examples with series

Function	Description	Source code
	Taylor series expansion	taylor.cpp

2.5 Source codes and output

2.5.1 Vector addition and subtraction

```
// "veca.cpp"
// Vector addition and subtraction
#include <iostream>
#include "symbolicc++.h"
using namespace std;

int main(void){
// 1. Block: Declaration of symbolic variables
 Symbolic x("x", 2), y("y", 2), z("z", 2), s("s", 2);

// 2. Block: Computations
 z = (x + y);
 s = (x - y);
// 3. Block: Output the results
cout<< "Addition z = " << z <<endl;
cout<< "Subtraction s = " << s <<endl;

system("PAUSE");
return 0;
}
/*
  Sample output:
  Addition z =
            [x0+y0]
            [x1+y1]

  Subtraction s =
            [x0-y0]
            [x1-y1]
*/
```

This symbolic computation gives directly the results of vector addition and subtraction for the given vectors

2.5.2 Scalar product

```
// "sp.cpp"
// Scalar product

#include <iostream>
#include "symbolicc++.h"
using namespace std;

int main(void){
// 1. Block: Declaration of symbolic variables
 Symbolic x("x", 2), y("y", 2), z("z", 2);

// 2. Block: Computations
 z = (x|y);

// 3. Block: Output the results
cout<< "z = " << z <<endl;

system("PAUSE");
return 0;
}
/*
  Sample output:
  z = x0*y0+x1*y1
*/
```

This symbolic computation gives directly the result of scalar product for the given expression

2.5.3 Cross product

```
// "cp.cpp"
// Cross product

#include <iostream>
#include "symbolicc++.h"
using namespace std;

int main(void){
// 1. Block: Declaration of symbolic variables
 Symbolic x("x", 3), y("y", 3), z("z", 3);

// 2. Block: Computations
 z = (x % y);

// 3. Block: Output the results
cout<< "z = " << z <<endl;

system("PAUSE");
return 0;
}
/*
  Sample output:
  z =
          [x1*y2-y1*x2]
          [y0*x2-x0*y2]
          [x0*y1-y0*x1]
*/
```

This symbolic computation gives directly the result of cross product for the given expression

2.5.4 Gradient

```
// "grad.cpp"
// Gradient example

#include <iostream>
#include "symbolicc++.h"
using namespace std;

int main(void){
// 1. Block: Declaration of symbolic variables
 Symbolic x("x"), y("y"), z("z");
 Symbolic f = x*x+y*y+z*z;
 Symbolic res("res", 3);
// 2. Block: Computations
res(0) = df(f,x);
res(1) = df(f,y);
res(2) = df(f,z);

// 3. Block: Output the results
cout<< "Gradient : "<< res <<endl;

system("PAUSE");
return 0;
}
/*
  Sample output:
Gradient :
          [2*x]
          [2*y]
          [2*z]
*/
```

This symbolic computation gives directly the result of gradient for the given expression

2.5.5 Divergence

```
// "diverge.cpp"
// Divergence

#include <iostream>
#include "symbolicc++.h"
using namespace std;

int main(void){
// 1. Block: Declaration of symbolic variables
 Symbolic x("x"), y("y"), z("z");
 Symbolic f = x*x+y*y+z*z;

// 2. Block: Computations
 Symbolic res = df(f,x) + df(f,y) + df(f,z);

// 3. Block: Output the results
cout<< "Divergence : "<< res <<endl;

system("PAUSE");
return 0;
}
/*
  Sample output:
Divergence : 2*x+2*y+2*z
*/
```

This symbolic computation gives directly the result of divergenceof the function above.

2.5.6 Differentiation

```
// "diff.cpp"
// Differentiation
```

```
#include <iostream>
#include "symbolicc++.h"

using namespace std;

int main(void){
// 1. Block: Declaration of symbolic variables
 Symbolic x("x");
 Symbolic y = sin(x) + cos(x);
// 2. Block: Computations
 Symbolic res = df(y,x);

// 3. Block: Output the results
cout<< "Differentiation : res = "<< res <<endl;

system("PAUSE");
return 0;
}
/*
  Sample output:
Differentiation : res = cos(x)-sin(x)
*/
```

The general terms after differentiation with symbolic computation are shown.

2.5.7 Integration

```
// "diff.cpp"
// Integration

#include <iostream>
#include "symbolicc++.h"

using namespace std;

int main(void){
// 1. Block: Declaration of symbolic variables
 Symbolic a("a"), t("t"), ;
 Symbolic f = exp(-a * t);

// 2. Block: Computations
 Symbolic res = integrate(f,t);

// 3. Block: Output the results
cout<< "Integration : res = "<< res <<endl;

system("PAUSE");
return 0;
}
/*
  Sample output:
Integration : res = -e^(-a*t)*a^(-1)
*/
```

The general terms after integration with symbolic computation are shown.

2.5.8 Multivariable polynomial differentiation

```
// "poly.cpp"
// Multivariable polynomial differentiation

#include <iostream>
#include "symbolicc++.h"
#include "multinomial.h"

using namespace std;

int main(void){
// 1. Block: Declaration of symbolic variables
 Multinomial <double>a("a");
 Multinomial <double>b("b");
 Multinomial <double>x("x");

 Multinomial <double> p = a*(x^3) + b*(x^2) + 1;
 // 2. Block: Computations

 Multinomial <double> res = Diff(p, "x");

 // 3. Block: Output the results
cout<< "Diff(p,x) => "<< res <<endl;

system("PAUSE");
return 0;
}
/*
  Sample output:
Diff(p,x) => (3)ax^2 + (2)bx
*/
```

This symbolic computation gives directly the result of multivariable polynomial differentiation for the given multinomial

2.5.9 Solution of Legendre differential equations

```
// legendre.cpp
// Legendre polynomial

#include <iostream>
#include "symbolicc++.h"
#include "legendre.h"

using namespace std;
int main(void){
int n=5;
  Symbolic x("x");
  Legendre P(n,x);

  // Calculate the first few Legendre polynomials
cout<< "P(0) = " << P <<endl;
for(int i=1;i<=n;i++)
  {
P.step();
cout<< "P("<< i << ") = " << P <<endl;
  }
cout<<endl;

  // Show that the Legendre differential equation is satisfied for n = 5

  Symbolic result;
result = df((1-x*x)*df(P.current(),x),x) + (n*(n+1))*P.current();
cout<< result <<endl;  // ==> 0
return 0;
}

/*
Sample output:
      P(0) = 1
      P(1) = x
      P(2) = 3/2*x^(2)-1/2
      P(3) = 5/2*x^(3)-3/2*x
      P(4) = 35/8*x^(4)-15/4*x^(2)+3/8
      P(5) = 63/8*x^(5)-35/4*x^(3)+15/8*x
```

```
0
*/
```

This symbolic computation shows the first few Legendre polynomials

2.5.10 Taylor series expansion

```
// taylor.cpp
// Taylor

#include <iostream>
#include "symbolicc++.h"

using namespace std;

int factorial(int N){
int result=1;
for(int i=2;i<=N;i++) result *= i;
return result;
}

int main(void){
int i, j, n=3;
  Symbolic u("u"), x("x"), result;
  Symbolic u0("",n), y("",n);
  u = u[x];
u0(0) = u*u+x;
for(j=1;j<n;j++) u0(j) = df(u0(j-1),x);

  // initial condition u(0)=1
u0(0) = u0(0)[u==1,x==0];

y(0) = u;
for(i=1;i<n;i++) y(i) = df(y(i-1),x);

  // substitution of initial conditions
for(i=1;i<n;i++)
     for(j=i;j>0;j--) u0(i) = u0(i)[y(j)==u0(j-1)];

for(i=0;i<n;i++) u0(i) = u0(i)[u == 1];

  // Taylor series expansion
result = 1;

for(i=0;i<n;i++)
result += (Symbolic(1)/factorial(i+1))*u0(i)*(x^(i+1));

cout<< "u(x) = " << result <<endl;
system("PAUSE");
return 0;
}

/*
Sample output
u(x) = x+3/2*x^(2)+4/3*x^(3)+1
*/
```

This symbolic computation gives directly the result of Taylor series expansion for the given series.

3. CONCLUSION

This paper gives brief information about symbolic computation techniques. The location of symbolic computation within programming language is illustrated.

Nowadays, imperative programming languages, such as C++, are generally used for numerical computations. The approach described in this paper can give the key idea to expand solution techniques.

Accordingly, symbolic computation techniques can be expanded to solve any sort of problems.

The general idea of this paper is to provide the implementation of well-known mathematical problems symbolically solved on the basis of C++ programming language. The properties of symbolic C++ have been illustrated using typical examples.

4. ACKNOWLEDGMENT

The authors would like to express gratitude to the research department of the University of Technology (em. Niyazi Ari, Prof. Dr. sch. Techn. ETH) Zurich, Switzerland.

5. REFERENCES

[1] Comparison of computer algebra systems http://en.wikipedia.org/wiki/Comparison_of_computer_algebra_systems

[2]Digital Math by Alphabet http://www.cs.ru.nl/~freek/digimath/xindex.html

[3] Mathematics in Open Project Directory http://www.dmoz.org/Science/Math/Software

[4] Combinatorial Software and Databases http://www.mat.univie.ac.at/~slc/divers/software.html

[5] Oberwolfach References on Mathematical Software http://orms.mfo.de/about

[6] The Scientific Computation System axiom http://axiom-developer.org/axiom-website/rosetta.html

[7] SymbolicC++: An Introduction to Computer Algebra using Object-Oriented Programming, ISBN-10: 1852332603

[8] Introduction to Symbolic C++ http://issc.uj.ac.za/symbolic/introsymb.pdf

[9] Samples of Symbolic C++ developers at web resource http://issc.uj.ac.za/symbolic/symbolic.html

[10] An open framework for symbolic computation within the C++ programming languagehttp://www.ginac.de

[11] “Applications of Symbolic computation in MATLAB” by Zhaparov M.K., Guvercin S.