stdafx.h

```cpp
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#pragma once
#include "targetver.h"

// TODO: reference additional headers your program requires here

#include <iostream>
#include <fstream>
#include <vector>
#include <valarray>
#include <algorithm>
#include <complex>
#include <cmath>

using namespace std;
using namespace std::complex_literals;
```

tools.h

```cpp
#pragma once

#include "stdafx.h"
```

```
class Tools
{
public:
	template<typename T>
	static valarray<T> LinSpace(T a, T b, size_t n=100)
	{
		valarray<T> t(n+1);
		T dt = (b - a) / n;
		t[0] = a;
		for (size_t i = 1; i < n; i++)
			t[i] = t[i-1] + dt;
		t[n] = b;
		return t;
	}

private:
	Tools()
	{
	}
};
```

table.h

```
#pragma once

#include "stdafx.h"

template <typename T>
class Table
{
public:
```

```
Table(valarray<T> x)
{
	data.push_back(x);
}

Table(valarray<T> x, valarray<T> y)
{
	data.push_back(x);
	data.push_back(y);
	jagged = x.size() != y.size();
}

void Add(valarray<T> x)
{
	data.push_back(x);
	if (!jagged)
		jagged = x.size() != data[0].size();
}

void SetColumnwise(bool value)
{
	columnwise = value;
}

bool GetColumnwise()
{
	return columnwise;
}

void Print(ostream& out=cout)
```

```
{
	size_t length = data.size();
	if (columnwise)
		if (jagged)
			out << "Columns must have equal length!" << endl;
		else
			for (size_t irow = 0; irow < data[0].size(); irow++)
			{
				for (size_t icol = 0; icol < length; icol++)
					out << data[icol][irow] << " ";
					out << endl;
			}
	else
		for (size_t icol = 0; icol < length; icol++)
		{
			for (size_t irow = 0; irow < data[icol].size(); irow++)
				out << data[icol][irow] << " ";
			out << endl;
		}
}

void Save(const char* filename)
{
	ofstream fout;
	fout.open(filename);
	Print(fout);
	fout.close();
	size_t length = data.size();
	fout.open("s.gpl");
	fout << "set yrange [-0.5:1.1]" << endl;
	fout << "plot ";
```

```
		for (size_t i = 1; i < length; i++)
		{
			fout << " 'tab' u 1:" << i + 1 << " w l";
			if (i == 1)
				fout << " t 'sin(x)/x'";
			else
				fout << " t 'T_{" << i - 2 << "}'";
			if (i < length - 1)
				fout << ", \\";
				fout << endl;
		}
		fout << "set yrange restore" << endl;
		fout << "pause -1" << endl;
		fout.close();
	}

	~Table()
	{
	}
private:
	vector<valarray<T>> data;
	bool columnwise = true;
	bool jagged = false;
};
```

Nm.cpp

```
// NM.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
```

```cpp
#include "Tools.h"
#include "Table.h"

typedef double Real;
typedef complex<Real> Complex;

Real f(Real x)
{
	if (x != 0)
		return sin(x) / x;
	else
		return 1;
}

Complex cf(Complex z)
{
	return exp(z);
}

void real_computing(valarray<Real>& t)
{
	auto x = cos(t);
	auto y = sin(t);
	Table<Real> tab(x, y);
	tab.Print();
}

void complex_computing(valarray<Real>& t)
{
	valarray<Complex> u(t.size());
	for (size_t i = 0; i < t.size(); i++)
```

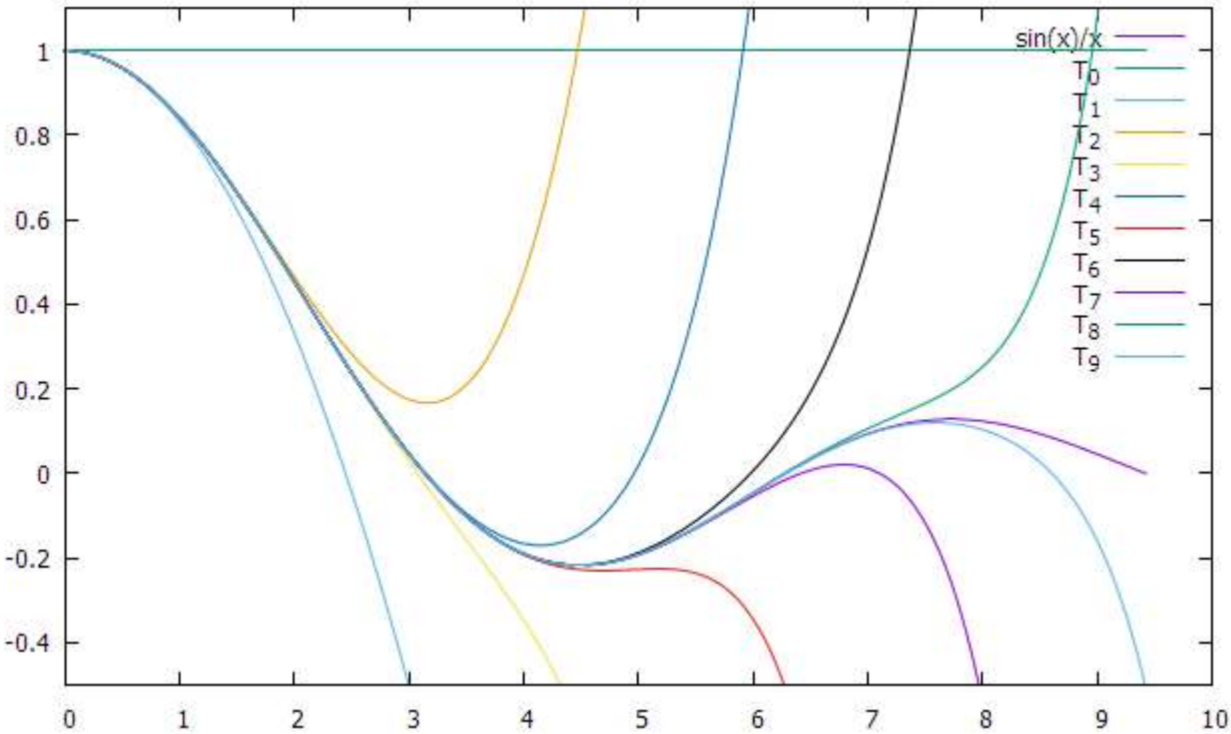
```
            u[i] = t[i] * 1i;
        auto z = u.apply(cf);
        Table<Complex> tab(u, z);
        tab.Print();
}

valarray<Real> Taylor(vector<Real> c, valarray<Real>& x)
{
        Real factorial = 1;
        for (size_t i = 2; i < c.size(); i++)
        {
                factorial *= i;
                c[i] /= factorial;
        }
        valarray<Real> T(x.size());
        for (int i = c.size() - 1; i > -1; i--) // Horner (Liu Hui 300 AC)
        {
                if (c[i] != 0)
                        T += c[i];
                if (i > 0)
                        T *= x;
        }
        return T;
}

int main()
{
        const Real PI = 3.141592653589793;
        size_t n = 300;
        auto t = Tools::LinSpace<Real>(0, 3*PI, n);
```

```
	//real_computing(t);
	//complex_computing(t);
	Table<Real> tab(t, t.apply(f));

	vector<Real> derivative;
	Real s = 1;
	for (size_t i = 0; i < 10; i++)
	{
		derivative.push_back(s/(2*i+1));
		derivative.push_back(0);
		s = -s;
		auto T = Taylor(derivative, t);
		tab.Add(T);
	}
	tab.Save("tab");
	return 0;
}
```



sin(x)/x
T_0
T_1
T_2
T_3
T_4
T_5
T_6
T_7
T_8
T_9
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
0
1
2
3
4
5
6
7
8
9
10