Project 1, SF2565

Hanna Hultin, TTMAM2
Mikael Persson, TTMAM2

September 13, 2016

Task 1

Consider the $2N$ degree Taylor polynomial for $\cos x$

$$
\begin{aligned}
\cos x \approx p(x) &= \sum_{n=0}^{N} (-1)^n \frac{x^{2n}}{(2n)!} \\
&= 1 + (-1)\frac{x^2}{2!} + (-1)^2\frac{x^4}{4!} + \cdots + (-1)^N \frac{x^{2N}}{(2N)!} \\
&= 1 - \frac{x \cdot x}{2 \cdot 1}\Big(1 - \frac{x \cdot x}{4 \cdot 3}\Big(1 - \dots \Big(1 - \frac{x \cdot x}{(2N)(2N-1)}\Big) \dots\Big).
\end{aligned}
$$

Hence, the polynomial may be evaluated backwords using the following scheme

$$
\begin{aligned}
b_N &= 1 - \frac{x \cdot x}{2N(2N-1)} \\
b_n &= 1 - \frac{x \cdot x}{2n(2n-1)} b_{n+1}, \quad n = N-1, N-2, \dots, 2, 1 \\
b_1 &= p(x).
\end{aligned}
$$

This is Horners' algorithm adjusted for the fact that each second term in the polynomial vanishes. Similarly for $\sin x$ the polynomial may be computed up to degree $2N+1$ by

$$
\begin{aligned}
b_N &= 1 - \frac{x \cdot x}{2N(2N+1)} \\
b_n &= 1 - \frac{x \cdot x}{2n(2n+1)} b_{n+1}, \quad n = N-1, N-2, \dots, 2, 1 \\
b_1 &= x \cdot p(x).
\end{aligned}
$$

```
for (int i=0; i<iterations;i++)
{
  do something
}
```

```cpp
/* Project 1 - Task 1, SF2565, KTH, 2016
 * comparison of taylor series for sin(x) and cos(x)
 * by cmath functions sin(x) and cos(x)
 * Hanna Hultin & Mikael Persson
 */


// libraries    ::      ::      ::      ::      ::

#include<iostream>
#include<cmath>
#include<iomanip>


// function declarations        ::      ::      ::

double sinTaylor(int N, double x);
double cosTaylor(int N, double x);
void errorBound(int N,double x,double sx,double sTx,double cx,double cTx);


// function definitions ::      ::      ::      ::

using namespace std;

int main(int argc, char *argv[])
{
  // main function. Prompts user for x and N. Calls sinTaylor and cosTaylor
  // functions and displays results of these compared to cmath functions.
  // The calls errorBound function to find if the error is bounded
  // by the N+1-term in the series.

  double sx, sTx, cx, cTx;      // sinx/cosx from cmath & Taylor polyn.
  double x; int N;
  cout << "x = "; cin >> x;     // Promt user for x and N
  cout << "N = "; cin >> N;

  sx = sin(x);
  cout << fixed << setprecision(15);    // Print more decimals
  cout << "cmath:   sin(x) = " << sx << endl;
  sTx = sinTaylor(N,x);
  cout << "Taylor:  sin(x) = " << sTx << endl;

  cx = cos(x);
  cout << "cmath:   cos(x) = " << cx << endl;
  cTx = cosTaylor(N,x);
  cout << "Taylor:  cos(x) = " << cTx << endl;

  errorBound(N,x,sx,sTx,cx,cTx);        // Compute and display errors etc

  return 0;
}

double sinTaylor(int N, double x)
{
  // returns value of N:th degree taylor polynomial for
  // sin function evaluated at x.
  // Horners algorithm is used to compute the series:
  // p(x) = a0 + a1*x + ... + aN*x^N
  // rewrite:
  // p(x) = a0 + x*(a1 + x*(a2 + ... + x*(aN-1 + x*aN)))...)
  // so we define a sequence:
  // bN = aN; bN-1 = aN-1 + bN*x; ... ; b0 = a0 + b1*x = p(x)
  //
```

```
  // Note, some adjustments are made to cope with the fact that each
  // second term vanishes in the sin series.

  double sinT;
  sinT = 1;
  for (int i = N; i > 0; i--)
  {
    sinT = 1-x*x*sinT/(double)(2*i*(2*i+1));
  }
  sinT = x*sinT;
  return sinT;
}

double cosTaylor(int N, double x)
{
  // returns value of N:th degree taylor polynomial for
  // cos function evaluated at x.
  // Horners algorithm is used.

  double cosT;
  cosT = 1;
  for (int i = N; i > 0; i--) // Analogous to sin
  {
    cosT = 1-x*x*cosT/(double)(2*i*(2*i-1));
  }
  return cosT;
}

void errorBound(int N,double x,double sx,double sTx,double cx,double cTx)
{
  // TODO, har jag gjort rÃtt hÃr?...

  double  sinTermN, cosTermN;
  sinTermN = sinTaylor(N+1,x) - sinTaylor(N,x);
  cosTermN = cosTaylor(N+1,x) - cosTaylor(N,x);

  cout << "   sin N+1-term = " << sinTermN << endl;
  cout << "   cos N+1-term = " << cosTermN << endl;
  cout << "sin err/N+1term = " << abs((sx-sTx)/sinTermN) << endl;
  cout << "cos err/N+1term = " << abs((cx-cTx)/cosTermN) << endl;

  // no return, fctn type void
}
```