LGS MF2 Computer Lab Session 1_Explicit

Finite-Difference Schemes for the Diffusion Equation with Smooth Initial Conditions

Schemes Investigated

In these first sessions we compare the accuracy of various difference schemes for solving the diffusion equation. This is the equation that arises when the Black-Scholes differential equation is transformed into a form suitable for treatment by finite-difference methods. We compare

(a) explicit finite-difference, with 400 time-steps (equivalent to the use of a binomial model, but on a grid rather than a tree);
(b) fully implicit, also with 400 time-steps;
(c) Crank-Nicholson, with 40 time-steps;
(d) Douglas, with 40 time-steps.

The solution method for type (a) is a simple updating rule, while (b), (c), (d) require the solution of tridiagonal systems of equations. This notebook and the associated C++ code look at case (a) above.

A Simple Test Problem with Smooth Initial Conditions

We consider the diffusion equation

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2} \tag{13.1}$$

on the region defined by

$$-2 \le x \le 2 \qquad \tau \ge 0 \tag{13.2}$$

The initial condition is

$$u(x, 0) = \sin\left(\frac{\pi x}{2}\right) \tag{13.3}$$

and the boundary conditions are

$$u(2, \tau) = u(-2, \tau) = 0 \tag{13.4}$$

This has the exact solution

$$u(x, \tau) = \sin\left(\frac{\pi x}{2}\right) e^{-\frac{\pi^2 \tau}{4}} \tag{13.5}$$

So it is a simple matter to test various difference schemes by comparing with this known exact solution. It should be emphasized that this type of smooth initial data, which also joins continuously onto the boundary conditions, is rather atypical of option-pricing problems. Our purpose here is to simplify matters to get a general feel for the relative merits of explicit and implicit scheme.

Explicit Scheme

Here are the price- and time-steps used, together with an output that is the value of the parameter $\alpha = \frac{\Delta\tau}{\Delta x^2}$.

```
dx = 0.025; dtau = 0.00025; alpha = dtau/dx^2
```

0.4

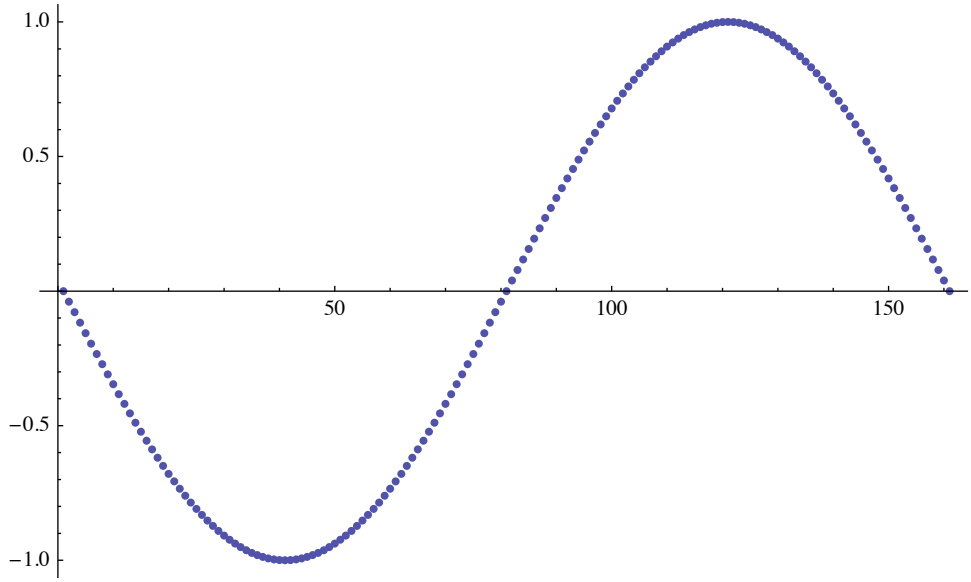
The number of time-steps is 400, and there are 160 space-steps.

```
M=400; nminus = 80; nplus = 80;
```

Here we set initial and boundary conditions, and plot the latter:

```
initial =
  Table[N[Sin[Pi * (k - 1 - nminus) / nminus] ], {k, 1, nminus + nplus + 1}];
lower = Table[0, {m, 1, M + 1}];
upper = Table[0, {m, 1, M + 1}];
```

```
ListPlot[initial]
```



Here we define a function to solve the problem on the given grid, the output is a list of values of u for the last three time points (for now we shall just use the final value). Note that this function makes use of the *Mathematica* vector compilation methods. The initial conditions, and upper and lower boundary conditions, are supplied as vectors, **initial**, **lower**, **upper**, as rank-1 objects that are real - note the syntax **{initial, _Real, 1}** to denote this. For further details see section 2.6.15 of the *Mathematica* book. Note that you can specify a vector (rank 1), matrix (rank 2) and higher order objects. No statement about dimension is needed - just the rank of the "tensor".

```
ExplicitSolver =
    Compile[
   {{initial, _Real, 1}, {lower, _Real, 1}, {upper, _Real, 1}, alpha},
        Module[{wold = initial, wnew = initial, wvold = initial,
     m, k, tsize = Length[lower], xsize = Length[initial]},
           For[m = 2, m <= tsize, m++,
                (wvold = wold; wold = wnew;
                    For[k = 2, k < xsize, k++,
                    (wnew[[k]] =
```

```
        alpha (wold[[k - 1]] + wold[[k + 1]]) + (1 - 2 alpha) wold[[k]])];
                    wnew[[1]] = lower[[m]];
                    wnew[[xsize]] = upper[[m]])];
        {wvold, wold, wnew}]
    ];
```

Now we apply this to get the solution at a time $\tau = 0.1$ (the units are irrelevant for our analysis):

```
Timing[soln = ExplicitSolver[initial, lower, upper, alpha];]
```

{0.047273, Null}

Now we do the interpolation to supply a continuous function:

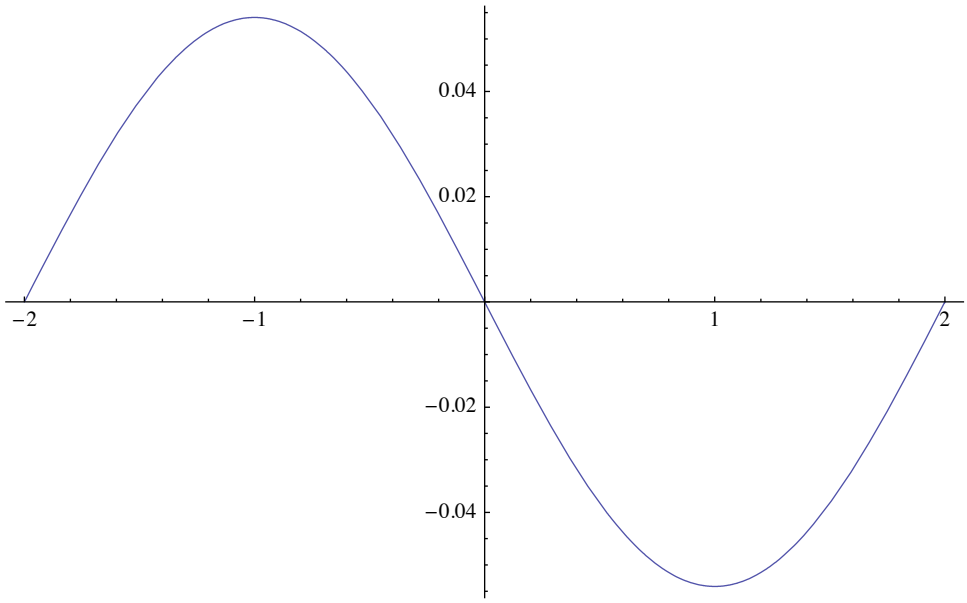
```
interpoldata =
Table[{(k-nminus-1)*dx ,soln[[3,k]]}, {k, 1, nminus+nplus+1}];
```

```
ufunc = Interpolation[interpoldata, InterpolationOrder -> 3]
```

InterpolatingFunction[(−2. 2.), <>]

Now we plot the error in the answer:

```
Plot[ufunc[x] - Sin[π x / 2] e^(1/4 (-π^2) 0.1`), {x, -2, 2}, PlotPoints → 50]
```



Finally we tabulate the numerical result, the exact result and the error in the numerical scheme. With 400 time-steps the error is manageably small.

```
samples = TableForm[Join[{{"x", "Explicit FD", "Exact", "Error"}},
Table[Map[PaddedForm[N[Chop[#1]],{5,6}]&,
N[{x,
ufunc[x],
Sin[Pi*x/2]*Exp[-(Pi^2 0.1)/4],
ufunc[x]- Sin[Pi*x/2]*Exp[-(Pi^2 0.1)/4]},5]],
{x, -2, 2, 0.25}]]]
```

```
x          Explicit FD  Exact      Error
-2.000000   0.000000     0.000000   0.000000
-1.750000  -0.298990    -0.299010   0.000013
-1.500000  -0.552470    -0.552490   0.000025
-1.250000  -0.721840    -0.721870   0.000032
-1.000000  -0.781310    -0.781340   0.000035
-0.750000  -0.721840    -0.721870   0.000032
-0.500000  -0.552470    -0.552490   0.000025
-0.250000  -0.298990    -0.299010   0.000013
 0.000000   0.000000     0.000000   0.000000
 0.250000   0.298990     0.299010  -0.000013
 0.500000   0.552470     0.552490  -0.000025
 0.750000   0.721840     0.721870  -0.000032
 1.000000   0.781310     0.781340  -0.000035
 1.250000   0.721840     0.721870  -0.000032
 1.500000   0.552470     0.552490  -0.000025
 1.750000   0.298990     0.299010  -0.000013
 2.000000   0.000000     0.000000   0.000000
```

C++ Model of Explicit Diffusion Test Problem

Go and look at code explicitdiffusion2.cpp....compile and run it for this case. The following tells *Mathematica* where this code is on my demonstration laptop. It would need changing on any other system!

```
SetDirectory["C:\Documents and
   Settings\William Shaw\My Documents\LGS0809\cppexamples"]
```

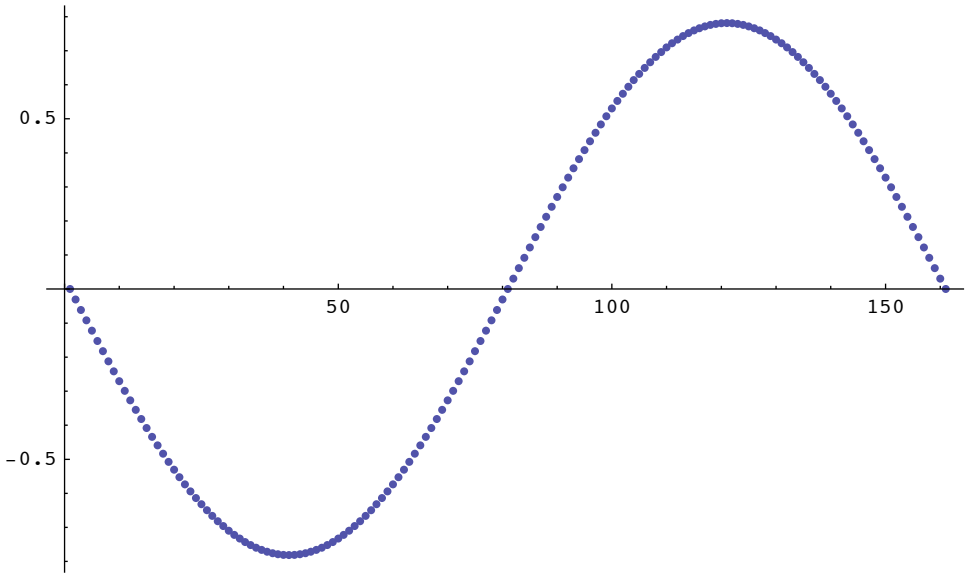
```
C:\Documents and Settings\William Shaw\My Documents\LGS0809\cppexamples
```

```
FileNames[]
```

```
{arithmeticone.cpp, arithmeticone.exe, arithmeticonefile.cpp,
 arithmeticonefile.exe, blackscholes.cpp, blackscholes.exe,
 bsoutput.txt, cumulativenormals.cpp, cumulativenormals.exe,
 dowhileiteration.cpp, ExplicitDiffusion2.cpp, ExplicitDiffusion2cpp.txt,
 ExplicitDiffusion2.exe, fdoutput.txt, foriteration.cpp,
 foriteration.exe, hello.cpp, hello.exe, helloname.cpp, helloname.exe,
 ifdeciding.cpp, ifdeciding.exe, incrementing.cpp, incrementing.exe,
 myoutput.txt, normaloutput.txt, realtypes.cpp, realtypes.exe,
 switchdeciding.cpp, switchdeciding.exe, whileiteration.cpp,
 whileiterationcruder.cpp, whileiterationcruder.exe, whileiteration.exe}
```

```
cppdata = ReadList["fdoutput.txt"];
```

```
ListPlot[cppdata]
```




```
Length[cppdata]
```

```
161
```

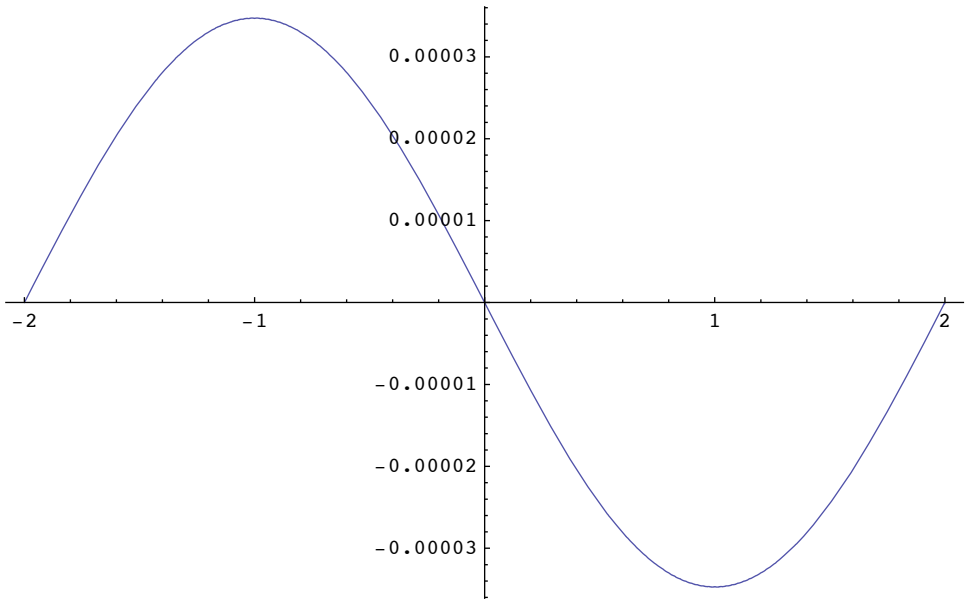
```
cppinterpoldata =
Table[{(k-nminus-1)*dx ,cppdata[[k]]}, {k, 1, nminus+nplus+1}];

cppfunc = Interpolation[cppinterpoldata, InterpolationOrder -> 3]
```

```
InterpolatingFunction[{{-2., 2.}}, <>]
```

Now we plot the error in the answer:

```
Plot[cppfunc[x] - Sin[π x / 2] e^(1/4 (-π^2) 0.1`), {x, -2, 2}, PlotPoints → 50]
```




```
mmadata = Transpose[interpoldata][[2]];

mmacppdiff = mmadata - cppdata;
```

```
Max[Abs[mmacppdiff]]
```

6.66134×10^{-16}

These are identical to the precision we have given. They should be!

Unstable Explicit Scheme

Let's double the time step and take α to 0.8. Looks innocuous enough!

```
dx = 0.025; dtau = 0.0005; alpha = dtau/dx^2
```

```
0.8
```

The number of time-steps is 200, and there are 160 space-steps.

```
M=200; nminus = 80; nplus = 80;
```

```
M * dtau
```

```
0.1
```

```
Timing[soln = ExplicitSolver[initial, lower, upper, alpha];]
```

```
{0.062, Null}
```

Now we do the interpolation to supply a continuous function:

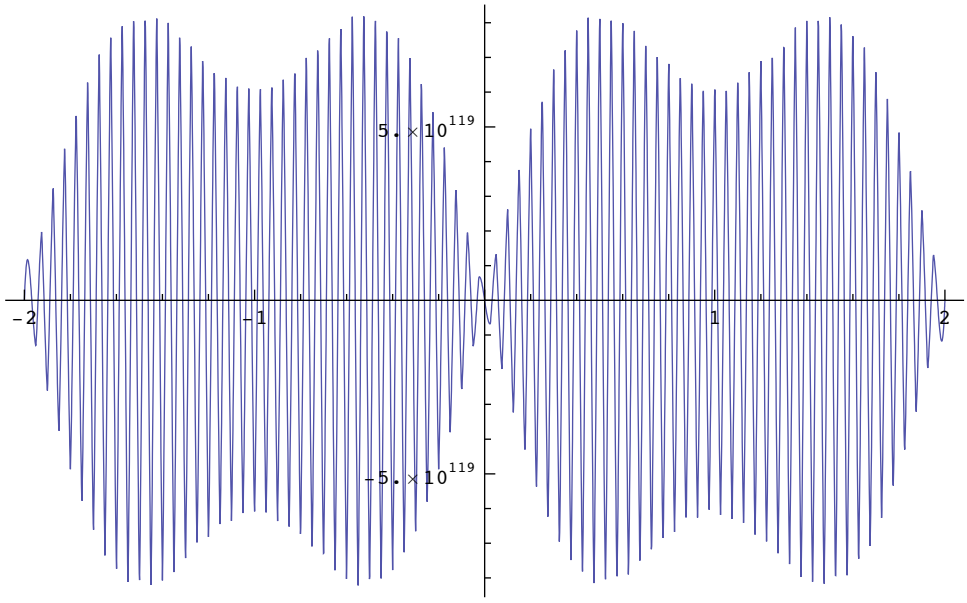
```
interpoldata =
Table[{(k-nminus-1)*dx ,soln[[3,k]]}, {k, 1, nminus+nplus+1}];
```

```
ufunc = Interpolation[interpoldata, InterpolationOrder -> 3]
```

```
InterpolatingFunction[{{-2., 2.}}, <>]
```

Now we plot the error in the answer: (NOTE THE AXES!!)

```
Plot[ufunc[x] - Sin[π x / 2] e^(1/4 (-π^2) 0.1`), {x, -2, 2}, PlotPoints → 50]
```



This is not a subtle matter. When you have instability, it is hard to miss, but you must do something to look for it. Now rerun the C++ code with M=200, N=80 and have a look at what happens.

```
cppdata = ReadList["fdoutput.txt", String];
cppdata[[Range[1, 10]]]
```

```
{0, 1.78013956766031e+051, -3.541704005783e+051, 5.26648036331699e+051,
 -6.93696882498808e+051, 8.53671173091579e+051, -1.00505909206946e+052,
 1.1465085057341e+052, -1.27684803113221e+052, 1.39510297406292e+052}
```

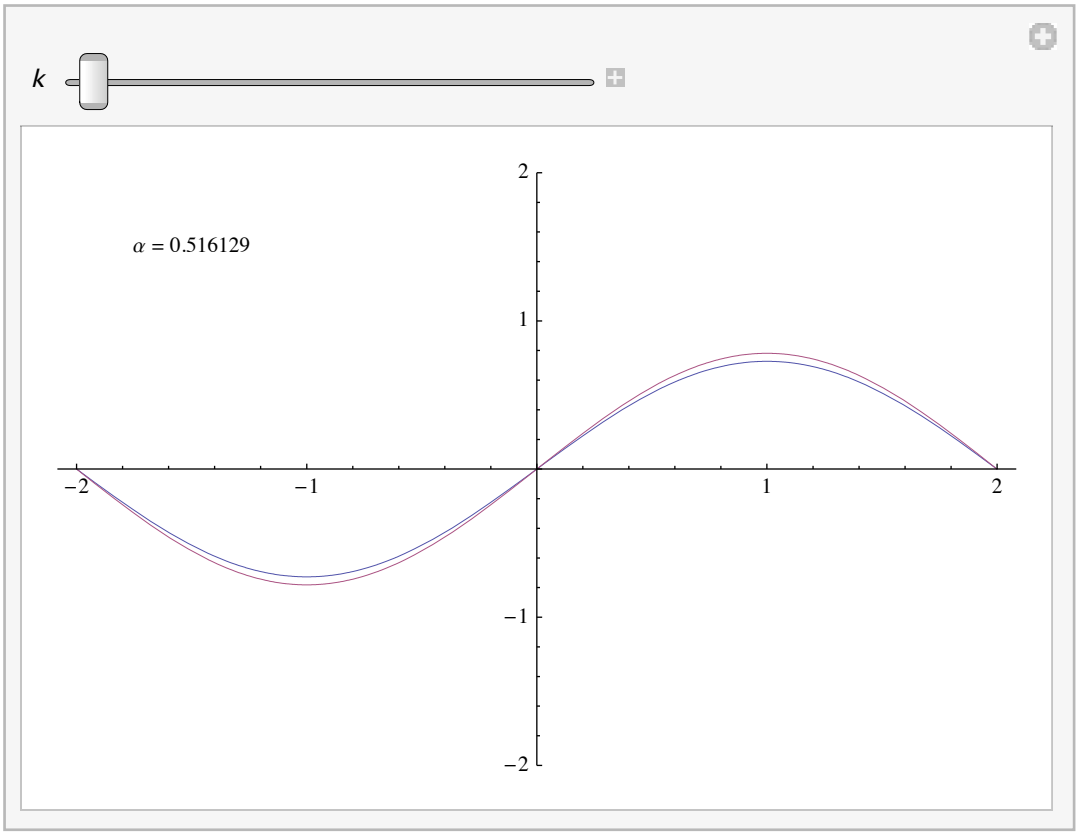
Stable to Unstable Transition

Let's take a look at the growth of instability.

■ Contrast M=310,309,...300

In this demonstration I will make an animation to show the transition to instability. This will use the **Manipulate** function (new in *Mathematica* 6). If you have an older version then replace **Manipulate** by **Do** and you will get a sequence of graphics showing the same effect.

```
Manipulate[(M = 310 - k;
  dx = 0.025; dtau = (320 / M) * 0.00125 / 4;
  nminus = 80; nplus = 80; alpha = dtau / dx^2;
  soln = ExplicitSolver[initial, lower, upper, alpha];
  interpoldata =
   Table[{(k - nminus - 1) * dx, soln[[3, k]]}, {k, 1, nminus + nplus + 1}];
  ufunc = Interpolation[interpoldata, InterpolationOrder → 3];
  Plot[{ufunc[x], Sin[Pi * x / 2] * Exp[-Pi^2 0.1 / 4]},
   {x, -2, 2}, PlotPoints → 50,
   PlotRange → {-2, 2}, PlotStyle →
    {Thickness[0.001], Thickness[0.001]}, Epilog → {Text[
      StringJoin["α = ", ToString[alpha]], {-1.5, 1.5}]}]), {k, 0, 7}]
```



Arrays in C++

This is an issue I am going to sidestep completely by using the public domain vector and matrix utilities created by the authors of Numerical Recipes in C++ [NRC++]. You might like to see the discussion of arrays in Section 1.2 of NRC++ under "Vectors and Matrices", as well as Appendix C of Joshi's book. Every C++ guru has probably written their own set. While I might have *Mathematica* guru status this does not remotely apply to C++ so I am going to work with the NRC++ set. Comments welcome on the relative merits of the schemes developed by NR, Joshi, Duffy and London etc.

[Now go to project with NR matrices and run that code - ExplicitDiffusion3]

```
SetDirectory["C:\Documents and
   Settings\William Shaw\My Documents\LGS0809\cppexamples"]
```

```
C:\Documents and Settings\William Shaw\My Documents\LGS0809\cppexamples
```

```
FileNames[]
```

```
{arithmeticone.cpp, arithmeticone.exe, arithmeticonefile.cpp,
 arithmeticonefile.exe, blackscholes.cpp, blackscholes.exe,
 bsoutput.txt, cumulativenormals.cpp, cumulativenormals.exe,
 dowhileiteration.cpp, ExplicitDiffusion2.cpp, ExplicitDiffusion2cpp.txt,
 ExplicitDiffusion2.exe, ExplicitDiffusion3.cpp,
 ExplicitDiffusion3cpp.txt, ExplicitDiffusion3.exe, fdoutput2.txt,
 fdoutput.txt, foriteration.cpp, foriteration.exe, hello.cpp,
 hello.exe, helloname.cpp, helloname.exe, ifdeciding.cpp,
 ifdeciding.exe, incrementing.cpp, incrementing.exe,
 myoutput.txt, normaloutput.txt, realtypes.cpp, realtypes.exe,
 switchdeciding.cpp, switchdeciding.exe, whileiteration.cpp,
 whileiterationcruder.cpp, whileiterationcruder.exe, whileiteration.exe}
```

```
cppdata2 = ReadList["fdoutput2.txt"];
```

```
mmacppdiff = mmadata - cppdata2;
```

```
Max[Abs[mmacppdiff]]
```

$$7.21645 \times 10^{-16}$$

So this is fine too.