This report describes using the free softwares


in place of Maple and Matlab to solve problem 3 concerning the construction of a Taylor integrating method which appeared on the Spring 2009 midterm of Math/CS 467/667 at the University of Nevada. The original solution using Maple may be found at

http://fractal.math.unr.edu/~ejolson/467-09/hw/mtans.pdf

for comparison.

1. Consider the initial value problem

\[
\begin{align*}
\{ \quad y' &= \cos(y) + \cos(x) \\
y(0) &= 0.
\end{align*}
\]

(i) Construct a 3rd-order Taylor method for solving this problem.

In general, a third order Taylor method has the update rule

\[
y_{n+1} = y_n + hf_0(x_n, y_n) + \frac{h^2}{2} f_1(x_n, y_n) + \frac{h^3}{3!} f_2(x_n, y_n)
\]

where

\[
f_i(x, y) = \frac{d^i}{dx^i} f(x, y).
\]

The Maxima script makejet.mac creates Fortran code for a Taylor method of order \(N\).

/* taylor.mac -- Create N-th order Taylor method for integrating
an ordinary differential equation.

Written 2010 by Eric Olson */

N:3$

eq:diff(y(x),x)=cos(y(x))+cos(x)$

genfort(x):=block ( 
  for y in x do 
    if atom(y) then 0 
    else if op(y)=":" then fortran(apply("=",args(y))) 
    else if op(y)="[" then genfort(y) 
    else fortran(y))$

print("Creating Taylor series method of order",N,"for")$
print(""")$
display(eq)"
Math/CS 467/667 Taylor Method with Open Source

The output of this script is

Creating Taylor series method of order 3 for

\[ \frac{dy}{dx} = \cos(y(x)) + \cos(x) \]

and the file \texttt{jet.f} generated by this script is

```
C Taylor order 3 time step for the differential equation
C
C 'diff(y(x),x,1) = \cos(y(x)) + \cos(x)
C
C Automatically generated on 02:49:53 Mon, 5/10/2010 (GMT+0)
C
```

```
Math/CS 467/667 Taylor Method with Open Source

```plaintext
function jet(x, yn, h)
  implicit real*8 (a-z)
  t1 = cos(x)
  t2 = cos(yn)
  t3 = t2 + t1
  t4 = sin(yn)
  t5 = -t3 * t4 - sin(x)
  Y1 = t3
  Y2 = t5
  Y3 = -t4 * t5 - t2 * t3**2 - t1
  jet = h * (h * (h * Y3 / 3.0E+0 + Y2) / 2.0E+0 + Y1) + yn
end
```

The Taylor integrater `taylor3a.c` can now be written as

```plaintext
#include <math.h>
#include "taylor3a.h"

extern double jet_(double *x, double *yn, double *h);

double taylor3a(double x0, double y0, double xn, int N){
  double h=(xn-x0)/N;
  double yn=y0;
  int n;
  for(n=0;n<N;n++){
    xn=x0+n*h;
    yn=jet_(&xn, &yn, &h);
  }
  return yn;
}
```

where `taylor3a.h` is given by

```plaintext
extern double taylor3a(double, double, double, int);
```

(ii) Compute $y(10)$ to 5 significant digits using your 3rd-order Taylor method.

The program for computing $y(10)$ is given by

```plaintext
#include <stdio.h>
#include <stdlib.h>
#include "taylor3a.h"

int main(){
  int N;
  printf("%5s %16s\n", "N", "y");
  for(N=8; N<=256; N*=2){
    double y=taylor3a(0, 0, 10, N);
    printf("%5d %16.5f\n", N, y);
  }
  return 0;
}
```
Math/CS 467/667 Taylor Method with Open Source

```c
printf("%5d %16.11f\n",N,y);
```

```c
exit(0);
```

This program can be built with the commands

```
gfortran -c jet.c
gcc -c taylor3a.c
gcc -o tayos3b tayos3b.c taylor3a.o jet.o -lm
```

and produces the output

<table>
<thead>
<tr>
<th>N</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.86457548562</td>
</tr>
<tr>
<td>16</td>
<td>0.86482318490</td>
</tr>
<tr>
<td>32</td>
<td>0.86411078690</td>
</tr>
<tr>
<td>64</td>
<td>0.86398264165</td>
</tr>
<tr>
<td>128</td>
<td>0.86396474966</td>
</tr>
<tr>
<td>256</td>
<td>0.86396241613</td>
</tr>
</tbody>
</table>

It is clear that $y(10) \approx 0.86396$ to 5 significant digits.

(iii) [*] Numerically verify the order of convergence of your code to be 3rd order.

Since this is a 3rd order method, theoretical expectations are that the error decreases by a factor of $2^3 = 8$ every time $n$ is doubled. The following C program computes an approximation for the exact answer using $n = 2^{15}$ and then checks from $n = 16, 32, \ldots, 1024$ that the error decreases by a factor of 8 each time.

```c
#include <stdio.h>
#include <stdlib.h>
#include "taylor3a.h"

const int M=8;

int main(){
    double y1[M];
    double yT=taylor3a(0,0,10,32768);
    int N1[M];
    int i,N;

    for(N=8,i=0;i<M;i++){
        N1[i]=N;
        y1[i]=taylor3a(0,0,10,N);
        N*=2;
    }

    printf("%5s %16s %20s %16s\n","N","y","error","ratio");
    { 
```
The output is

<table>
<thead>
<tr>
<th>N</th>
<th>y</th>
<th>error</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>0.86482318490</td>
<td>8.611087197790e-04</td>
<td>0.712348420285</td>
</tr>
<tr>
<td>32</td>
<td>0.86411078690</td>
<td>1.487107152023e-04</td>
<td>5.790495450228</td>
</tr>
<tr>
<td>64</td>
<td>0.86398264165</td>
<td>2.056547146689e-05</td>
<td>7.231087088943</td>
</tr>
<tr>
<td>128</td>
<td>0.86396474966</td>
<td>2.673474348347e-06</td>
<td>7.692413985423</td>
</tr>
<tr>
<td>256</td>
<td>0.86396241613</td>
<td>3.399449226560e-07</td>
<td>7.864433824924</td>
</tr>
<tr>
<td>512</td>
<td>0.86396211901</td>
<td>4.283216037404e-08</td>
<td>7.936674678264</td>
</tr>
<tr>
<td>1024</td>
<td>0.86396208156</td>
<td>5.374421574444e-09</td>
<td>7.969631667473</td>
</tr>
</tbody>
</table>

The fact that the list of ratios given in the last column of the above table is close to 8 verifies the method is converging with third order.