Daniel Purcell
MATH 4670
December 7, 2005

5.2 Problem 1.a

Use Euler's method to approximation solutions for

$$y' = te^{3t} - 2y, \text{ for } 0 \le t \le 1, \text{ with } y(0) = 0 \text{ and } h = 0.5$$

C Code

```
#include <stdio.h>
#include <math.h>


double f(double t, double w);

int main()
{
   int n = 30, i;
   double w = 0, a = 0, b = 1, t, h;
   h = (b - a) / (double) n;
   for (i = 0; i <= n; i++)
   {
      t = a + i * h;
      w = w + h * f(t, w);
      printf("%d \t %1.8f \t %3.8f \t \n", i, t, w);
   }

   return 0;
}

double f(double t, double w)
{
   return t * exp(3 * t) - 2 * w;
}
```

Results

```
i              t               w
-----------------------------------
0   0.00000000   0.00000000
1   0.05000000   0.00290459
2   0.10000000   0.00936342
3   0.15000000   0.02018942
```

```
4    0.20000000    0.03639167
5    0.25000000    0.05921500
6    0.30000000    0.09018755
7    0.35000000    0.13117769
8    0.40000000    0.18446226
9    0.45000000    0.25280811
10    0.50000000    0.33956952
11    0.55000000    0.44880451
12    0.60000000    0.58541349
13    0.65000000    0.75530448
14    0.70000000    0.96558998
15    0.75000000    1.22482108
16    0.80000000    1.54326603
17    0.85000000    1.93324133
18    0.90000000    2.40950513
19    0.95000000    2.98972425
20    1.00000000    3.69502867
```

5.2 Problem 1.c

Use Euler's method to approximate the solutions for

$$y' = 1 + (y/t), \text{ for } 0 \leq t \leq 1, \text{ with } y(0) = 1$$

C Code

```c
#include <stdio.h>
#include <math.h>


double f(double t, double w);

int main()
{
   int n = 20, i;
   double w = 2, a = 1, b = 2, t, h;
   h = (b - a) / (double) n;
   for (i = 0; i <= n; i++)
   {
      t = a + i * h;
      w = w + h * f(t, w);
      printf("%d \t %1.8f \t %3.8f \t \n", i, t, w);
   }

   return 0;
}
```

```
double f(double t, double w)
{
   return 1 + (w/t);
}
```

Results

```
i            t               w
-----------------------------------
0   1.00000000   2.15000000
1   1.05000000   2.30238095
2   1.10000000   2.45703463
3   1.15000000   2.61386222
4   1.20000000   2.77277315
5   1.25000000   2.93368408
6   1.30000000   3.09651808
7   1.35000000   3.26120393
8   1.40000000   3.42767550
9   1.45000000   3.59587121
10   1.50000000   3.76573358
11   1.55000000   3.93720886
12   1.60000000   4.11024664
13   1.65000000   4.28479957
14   1.70000000   4.46082308
15   1.75000000   4.63827517
16   1.80000000   4.81711615
17   1.85000000   4.99730848
18   1.90000000   5.17881659
19   1.95000000   5.36160676
20   2.00000000   5.54564693
```

5.2 Problem 1.d

Use Euler's method to approximate the solutions for

$$y' = \cos 2t + \sin 3t, \text{ for } 0 \leq t \leq 1, \text{ with } y(0) = 1$$

C Code

```
#include <stdio.h>
#include <math.h>


double f(double t, double w);
```

```
int main()
{
   int n = 20, i;
   double w = 1, a = 0, b = 1, t, h;
   h = (b - a) / (double) n;
   for (i = 0; i <= n; i++)
   {
      t = a + i * h;
      w = w + h * f(t, w);
      printf("%d \t %1.8f \t %3.8f \t \n", i, t, w);
   }

   return 0;
}

double f(double t, double w)
{
   return cos(2*t) + sin(3*t);
}
```

Results

```
i            t               w
-----------------------------------
0   0.00000000   1.05000000
1   0.05000000   1.10722211
2   0.10000000   1.17100145
3   0.15000000   1.24051656
4   0.20000000   1.31480173
5   0.25000000   1.39276279
6   0.30000000   1.47319592
7   0.35000000   1.55480919
8   0.40000000   1.63624648
9   0.45000000   1.71611315
10   0.50000000   1.79300301
11   0.55000000   1.86552607
12   0.60000000   1.93233634
13   0.65000000   1.99215927
14   0.70000000   2.04381809
15   0.75000000   2.08625861
16   0.80000000   2.11857179
17   0.85000000   2.14001376
18   0.90000000   2.15002264
19   0.95000000   2.14823207
20   1.00000000   2.13448073
```

5.2 Problem 8.e

Use Taylor's method of order 4 with $h = 0.1$ to approximate

$$y' = \frac{2}{t}y + t^2 e^t, 1 \le t \le 2, y(1) = 0$$

with the exact solution

$$y(t) = t^2(e^t - e)$$

C Code

```
#include <stdio.h>
#include <math.h>


double f(double t, double w);
double y2(double t, double y);
double y3(double t, double y);
double exact(double t);
double taylor(double t, double y, double h);

int main()
{
   int n = 25, i;
   double w = 0, a = 1, b = 2, t, h;
   h = (b - a) / (double) n;
   for (i = 0; i <= n; i++)
   {
      t = a + i * h;
      w = w + h * taylor(t, w, h);
      printf("%d \t %1.8f \t %3.8f \t %3.8f \t %3.8f \n", i, t, w, exact(t),
                      exact(t) - w);
   }

   return 0;
}

double f(double t, double w)
{
   return 2/t * w + t*t * exp(t);
}

double y2(double t, double y)
{
   return (-2/(t * t)) * y + 2 * t * exp(t) + t * t * exp(t) + 2 * t * f(t, y);
}
```

```
double y3(double t, double y)
{
  return (-4/(t*t*t) * y + 2*exp(t) + t*t*exp(t)) + (-2/(t*t) * f(t, y))
  + f(t, y) * 2 + 2 * t * y2(t, y);
}

double exact(double t)
{
   return t * t * (exp(t) - exp(1));
}

double taylor(double t, double y, double h)
{
   return f(t, y) + h/2 * y2(t, y) + (h*h*h)/6 * y3(t, y);
}
```

Results

i	t	w	y(t)	error
0	1.00000000	0.11961948	0.00000000	-0.11961948
1	1.04000000	0.26369547	0.11998750	-0.14370798
2	1.08000000	0.43488933	0.26407030	-0.17081902
3	1.12000000	0.63605321	0.43474039	-0.20131282
4	1.16000000	0.87024209	0.63465419	-0.23558791
5	1.20000000	1.14072644	0.86664254	-0.27408390
6	1.24000000	1.45100557	1.13372112	-0.31728445
7	1.28000000	1.80482194	1.43910158	-0.36572036
8	1.32000000	2.20617611	1.78620315	-0.41997296
9	1.36000000	2.65934269	2.17866506	-0.48067762
10	1.40000000	3.16888718	2.62035955	-0.54852763
11	1.44000000	3.73968379	3.11540565	-0.62427814
12	1.48000000	4.37693427	3.66818370	-0.70875057
13	1.52000000	5.08618786	4.28335075	-0.80283710
14	1.56000000	5.87336236	4.96585672	-0.90750563
15	1.60000000	6.74476643	5.72096153	-1.02380491
16	1.64000000	7.70712318	6.55425311	-1.15287007
17	1.68000000	8.76759506	7.47166654	-1.29592852
18	1.72000000	9.93381022	8.47950405	-1.45430617
19	1.76000000	11.21389031	9.58445628	-1.62943403
20	1.80000000	12.61647998	10.79362466	-1.82285532
21	1.84000000	14.15077790	12.11454498	-2.03623292
22	1.88000000	15.82656961	13.55521229	-2.27135732
23	1.92000000	17.65426228	15.12410717	-2.53015511
24	1.96000000	19.64492128	16.83022338	-2.81469790

```
25   2.00000000   21.81030898   18.68309708   -3.12721190
```

The results become less accurate as t increases.

5.3 Problem 1.b

Use the Midpoint method to approximate the solution for

$$y' = te^{3t} - 2y, 0 \leq t \leq 1, y(0) = 0$$

and compare with the actual solution

$$y(t) = t + 1/(1 - t)$$

C Code

```
#include <stdio.h>
#include <math.h>


double f(double t, double w);
double exact(double t);

int main()
{
   int n = 20, i;
   double w = 1, a = 2, b = 3, t, h;
   h = (b - a) / (double) n;
   for (i = 0; i <= n; i++)
   {
      t = a + i * h;
      w = w + h * f(t + .5 * h, w + h * .5 * f(t, w));
      printf("%d \t %1.8f \t %3.8f \t %3.8f \t %3.8f \n", i, t, w, exact(t),
                                                              exact(t) - w);
   }

   return 0;
}

double f(double t, double w)
{
   return 1.0 + (t - w)*(t - w);
}

double exact(double t)
{
```

```
  return t + 1.0/(1.0 - t);
}
```

Results

i	t	w	y(t)	error
0	2.00000000	1.09753125	1.00000000	-0.09753125
1	2.05000000	1.19075661	1.09761905	-0.09313757
2	2.10000000	1.28023492	1.19090909	-0.08932583
3	2.15000000	1.36643244	1.28043478	-0.08599766
4	2.20000000	1.44974129	1.36666667	-0.08307463
5	2.25000000	1.53049357	1.45000000	-0.08049357
6	2.30000000	1.60897239	1.53076923	-0.07820316
7	2.35000000	1.68542062	1.60925926	-0.07616137
8	2.40000000	1.76004775	1.68571429	-0.07433347
9	2.45000000	1.83303544	1.76034483	-0.07269061
10	2.50000000	1.90454197	1.83333333	-0.07120864
11	2.55000000	1.97470593	1.90483871	-0.06986722
12	2.60000000	2.04364913	1.97500000	-0.06864913
13	2.65000000	2.11147909	2.04393939	-0.06753970
14	2.70000000	2.17829109	2.11176471	-0.06652638
15	2.75000000	2.24416982	2.17857143	-0.06559839
16	2.80000000	2.30919086	2.24444444	-0.06474641
17	2.85000000	2.37342183	2.30945946	-0.06396237
18	2.90000000	2.43692343	2.37368421	-0.06323922
19	2.95000000	2.49975031	2.43717949	-0.06257082
20	3.00000000	2.56195178	2.50000000	-0.06195178

As t increases, the results become more accurate.