

Math 660: Problem Set 5

Matthew Grasinger

April 24, 2017

1 C1: ADI

The relative L_∞ errors for $k = h = \frac{1}{10}$, $\frac{1}{20}$ and $\frac{1}{40}$ were 0.00163, 0.000476, and 0.000161, respectively. This means that, roughly, each time k and h were halved the error decreased by a factor of four. This suggests that the approximation accuracy is second order, which agrees with the theory. In the three figures that follow, the ADI approximation is compared with the exact solution. The approximation agrees well with the exact solution in all cases.

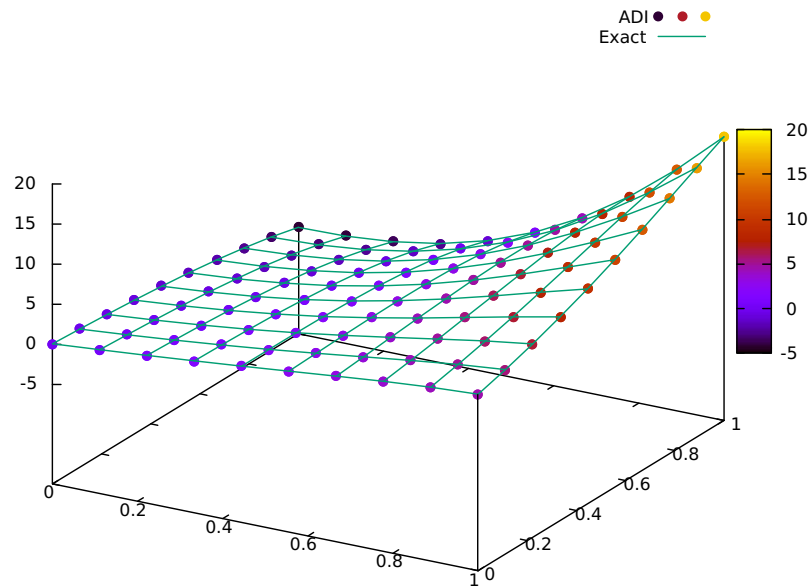


Figure 1: ADI approximation compared with exact solution. $h = \frac{1}{10}$.

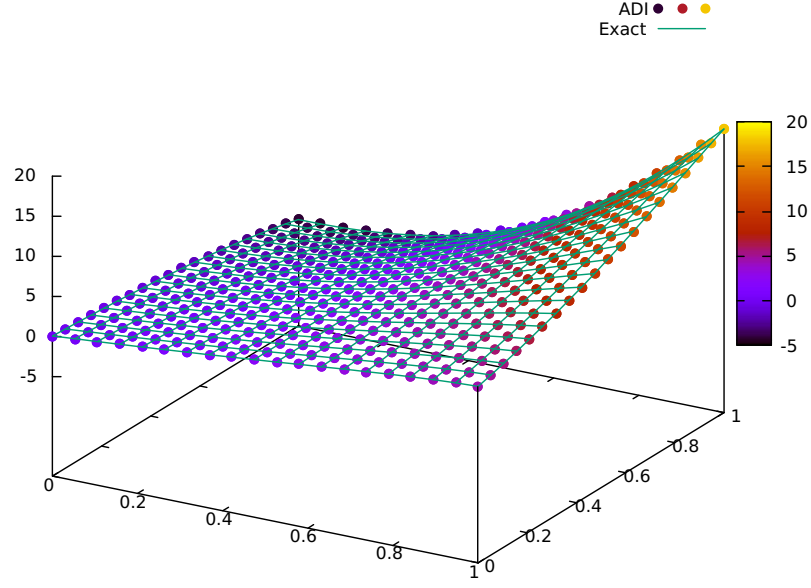


Figure 2: ADI approximation compared with exact solution. $h = \frac{1}{20}$.

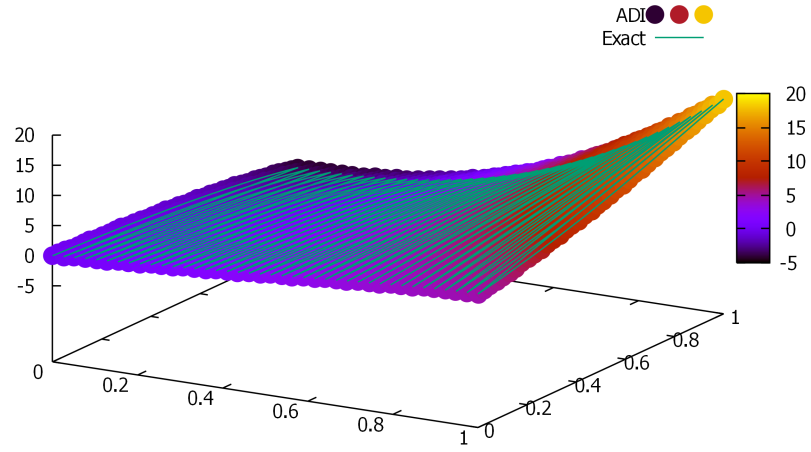


Figure 3: ADI approximation compared with exact solution. $h = \frac{1}{40}$.

1.1 Source Code

```

1  asoln(x, y, t) = exp(0.75 * t) * sin(2 * x - y) * cosh(1.5 * (x + y));
2
3  errs = [];
4  hs = [1/10; 1/20; 1/40];
5
6  for h in hs
7      k = h;
8       $\mu$  = k / (h*h);
9      println("k = $k, h = $h");
10
11     const aax = - $\mu$  / 2;
12     const bbx = ( $\mu$  + 1);
13     const ccx = aax;
14
15     const aay = - $\mu$ ;
16     const bby = (2 *  $\mu$  + 1);
17     const ccy = aay;
18
19     xs = linspace(0.0, 1.0, Int(round(1.0 / h)));
20     ys = copy(xs);
21     ts = linspace(0.0, 1.0, Int(round(1.0 / k)));
22     const M, L, K = length(xs), length(ys), length(ts);
23
24     u = zeros(M, L, K);
25     for (m, x) in zip(1:M, xs), (l, y) in zip(1:L, ys)
26         u[m, l, 1] = asoln(x, y, 0);
27     end
28
29     for n in 1:K-1
30         u_temp = zeros(M, L);
31         thalf = (ts[n]+ts[n+1]) / 2;
32         # calculate boundary terms
33         for l in 1:L
34             u_temp[1, l] = asoln(0.0, ys[l], thalf);
35             u_temp[M, l] = asoln(1.0, ys[l], thalf);
36             u[1, l, n+1] = asoln(0.0, ys[l], ts[n+1]);
37             u[M, l, n+1] = asoln(1.0, ys[l], ts[n+1]);
38         end
39         for m in 2:M-1
40             u_temp[m, 1] = asoln(xs[m], 0.0, thalf);
41             u_temp[m, L] = asoln(xs[m], 1.0, thalf);
42             u[m, 1, n+1] = asoln(xs[m], 0.0, ts[n+1]);
43             u[m, L, n+1] = asoln(xs[m], 1.0, ts[n+1]);
44         end
45
46         for l in 2:L-1
47             # calculate pi and qi for Thomas' algorithm
48             p = zeros(L);
49             q = zeros(L);
50             p[2], q[2] = 0.0, asoln(0.0, ys[l], thalf);
51             for m=2:M-1
52                 dd = u[m, l, n] + ( $\mu$  *
53                     (u[m, l+1, n] - 2 * u[m, l, n] + u[m, l-1, n]));
54                 denom = aax * p[m] + bbx;
55                 p[m+1] = -ccx / denom;
56                 q[m+1] = (dd - aax * q[m]) / denom;
57             end
58             u_temp[M, l] = asoln(1.0, ys[l], thalf);
59             for m=M-1:-1:2
60                 u_temp[m, l] = p[m+1] * u_temp[m+1, l] + q[m+1];
61             end
62             u_temp[1, l] = asoln(0.0, ys[l], thalf);
63         end
64     end

```

```

65     for m in 2:M-1
66         # calculate pi and qi for Thomas' algorithm
67         p = zeros(M);
68         q = zeros(M);
69         p[2], q[2] = 0.0, asoln(xs[m], 0.0, ts[n+1]);
70         for l=2:L-1
71             dd = u_temp[m, l] + (mu / 2 *
72                 (u_temp[m+1, l] - 2 * u_temp[m, l] + u_temp[m-1, l]));
73             denom = aay * p[l] + bby;
74             p[l+1] = -ccy / denom;
75             q[l+1] = (dd - aay * q[l]) / denom;
76         end
77         u[m, L, n+1] = asoln(xs[m], 1.0, ts[n+1]);
78         for l=L-1:-1:2
79             u[m, l, n+1] = p[l+1] * u[m, l+1, n+1] + q[l+1];
80         end
81         u[m, 1, n+1] = asoln(xs[m], 0.0, ts[n+1]);
82     end
83
84     u_exact = zeros(M, L);
85     for m in 1:M, l in 1:L
86         u_exact[m, l] = asoln(xs[m], ys[l], ts[n+1]);
87     end
88
89     if n % 5 == 0
90         println("t=$(ts[n+1]), relative L $\infty$  error: ", norm(u[:, :, n+1] - u_exact,
91             Inf) / norm(u_exact, Inf));
92         println("t=$(ts[n+1]), relative L2 error: ", norm(u[:, :, n+1] - u_exact,
93             2) / norm(u_exact, 2));
94         open(w -> begin
95             for m in 1:M, l in 1:L
96                 write(w, "$(xs[m]),$(ys[l]),$(u[m, l, n+1]),$(asoln(xs[m], ys[l], ts[n
97                     +1]))\n");
98             end
99         end, "h-$(Int(round(h*100)))_t-$(Int(round(ts[n+1]*100))).csv", "w");
100
101     u_exact = zeros(M, L);
102     for m in 1:M, l in 1:L
103         u_exact[m, l] = asoln(xs[m], ys[l], ts[K]);
104     end
105     push!(errs, maximum(map(x -> abs(x), u[:, :, K] - u_exact)));
106     println("t=1.0, relative L $\infty$  error: ", norm(u[:, :, K] - u_exact, Inf) / norm
107         (u_exact, Inf));
108     println("t=1.0, relative L2 error: ", norm(u[:, :, K] - u_exact, 2) / norm
109         (u_exact, 2));
110     println();
111     open(w -> begin
112         for m in 1:M, l in 1:L
113             write(w, "$(xs[m]),$(ys[l]),$(u[m, l, K]),$(asoln(xs[m], ys[l], ts[K]))
114                 \n");
115         end
116     end, "h-$(Int(round(h*100)))_end.csv", "w");
117
118     println(@sprintf("%10s %10s %10s", "h", "max(|e|)", "ratio"));
119     println(@sprintf("%10.4lf %10.4lf %10s", hs[1], errs[1], "N/A"));
120     println(@sprintf("%10.4lf %10.4lf %10lf", hs[2], errs[2], errs[1]/errs[2]));
121     println(@sprintf("%10.4lf %10.4lf %10lf", hs[3], errs[3], errs[2]/errs[3]));

```

2 Douglas-Rachford (Optional)

For the Douglas-Rachford scheme the grid spacings were $h = \frac{1}{10}, \frac{1}{20}$ and $\frac{1}{40}$, but k was taken to be $k = h^2$ because the Douglas-Rachford scheme is second order accurate in space but only first order accurate in time. The relative L_∞ errors for $h = \frac{1}{10}, \frac{1}{20}$ and $\frac{1}{40}$ were 0.00114, 0.000393, and 0.000314, respectively. Notice that, roughly, when h was halved the first time (from $h = \frac{1}{10}$ to $h = \frac{1}{20}$ and $k = \frac{1}{100}$ to $k = \frac{1}{400}$) the error decreased by a factor of four, which agrees with the theory. However, when h was halved again there were diminishing returns on the accuracy as the accuracy only increased by less than 25%. In the three figures that follow, the ADI approximation is compared with the exact solution. The approximation agrees well with the exact solution in all cases.

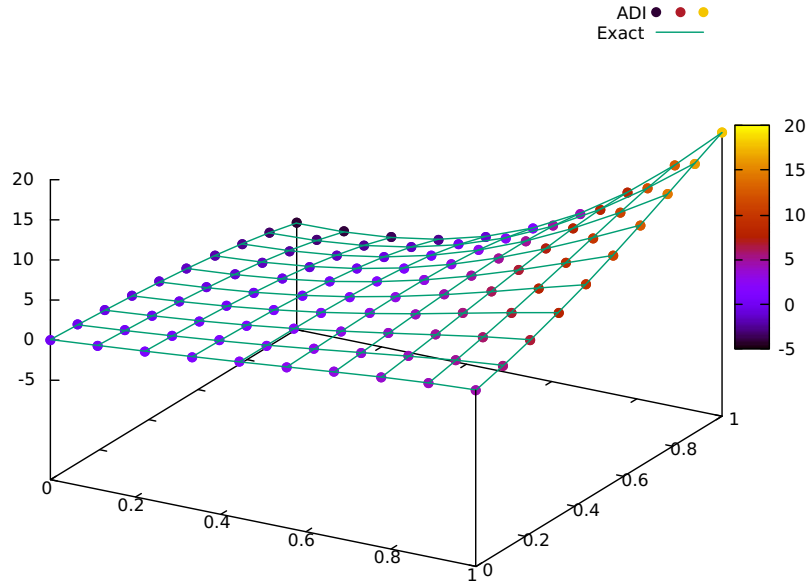


Figure 4: ADI approximation compared with exact solution. $h = \frac{1}{10}$, $k = \frac{1}{100}$.

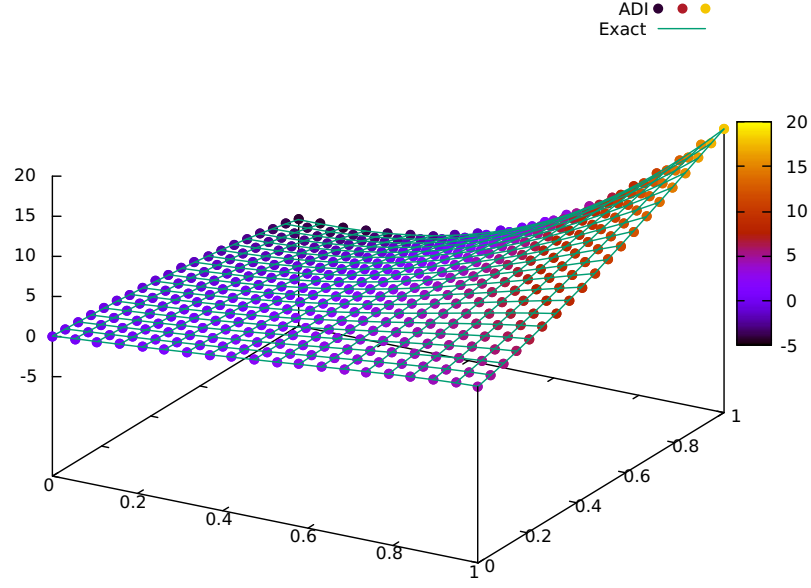


Figure 5: ADI approximation compared with exact solution. $h = \frac{1}{20}$, $k = \frac{1}{400}$.

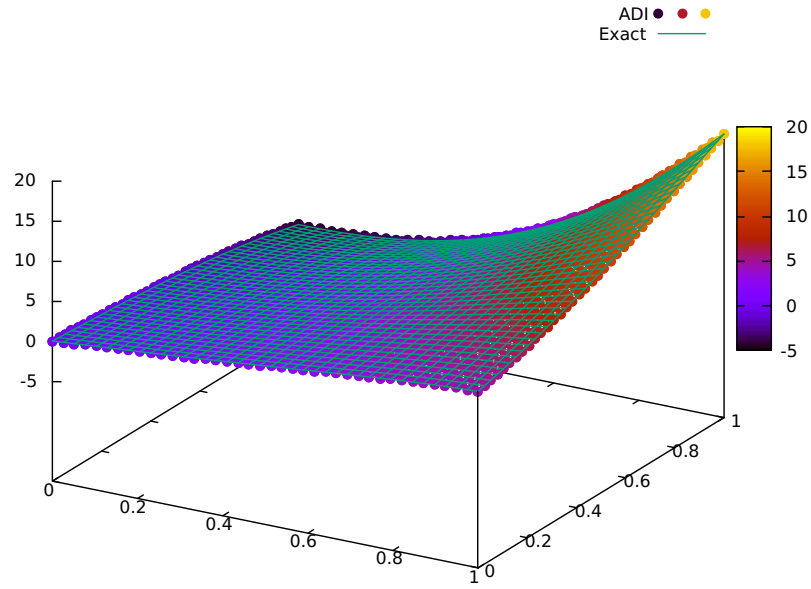


Figure 6: ADI approximation compared with exact solution. $h = \frac{1}{40}$, $k = \frac{1}{1600}$.

2.1 Source Code

```

1  dfilename(h, t) = @sprintf("h-%04d_t-%04d.csv", Int(round(h*100)), Int(round
   (t*100)));
2  dfilename(h) = @sprintf("h-%04d_end.csv", Int(round(h*100)));
3
4  asoln(x, y, t) = exp(0.75 * t) * sin(2 * x - y) * cosh(1.5 * (x + y));
5
6  errs = [];
7  hs = [1/10; 1/20; 1/40];
8
9  for h in hs
10     k = h^2;
11     output_step = Int(round(1 / k * 0.2));
12
13     μ = k / (h*h);
14     println("k = $k, h = $h");
15
16     const aax = -μ;
17     const bbx = (2 * μ + 1);
18     const ccx = aax;
19
20     const aay = -2 * μ;
21     const bby = (4 * μ + 1);
22     const ccy = aay;
23
24     xs = linspace(0.0, 1.0, Int(round(1.0 / h)));
25     ys = copy(xs);
26     ts = linspace(0.0, 1.0, Int(round(1.0 / k)));
27     const M, L, K = length(xs), length(ys), length(ts);
28
29     u = zeros(M, L, K);
30     for (m, x) in zip(1:M, xs), (l, y) in zip(1:L, ys)
31         u[m, l, 1] = asoln(x, y, 0);
32     end
33
34     for n in 1:K-1
35         u_temp = zeros(M, L);
36         thalf = (ts[n]+ts[n+1]) / 2;
37         # calculate boundary terms
38         for l in 1:L
39             u_temp[1, l] = asoln(0.0, ys[l], thalf);
40             u_temp[M, l] = asoln(1.0, ys[l], thalf);
41             u[1, l, n+1] = asoln(0.0, ys[l], ts[n+1]);
42             u[M, l, n+1] = asoln(1.0, ys[l], ts[n+1]);
43         end
44         for m in 2:M-1
45             u_temp[m, 1] = asoln(xs[m], 0.0, thalf);
46             u_temp[m, L] = asoln(xs[m], 1.0, thalf);
47             u[m, 1, n+1] = asoln(xs[m], 0.0, ts[n+1]);
48             u[m, L, n+1] = asoln(xs[m], 1.0, ts[n+1]);
49         end
50
51         for l in 2:L-1
52             # calculate pi and qi for Thomas' algorithm
53             p = zeros(L);
54             q = zeros(L);
55             p[2], q[2] = 0.0, asoln(0.0, ys[l], thalf);
56             for m=2:M-1
57                 dd = u[m, l, n] + (2 * μ *
58                     (u[m, l+1, n] - 2 * u[m, l, n] + u[m, l-1, n]));
59                 denom = aax * p[m] + bbx;
60                 p[m+1] = -ccx / denom;
61                 q[m+1] = (dd - aax * q[m]) / denom;
62             end
63             u_temp[M, l] = asoln(1.0, ys[l], thalf);

```



```

64     for m=M-1:-1:2
65         u_temp[m, l] = p[m+1] * u_temp[m+1, l] + q[m+1];
66     end
67     u_temp[1, l] = asoln(0.0, ys[l], thalf);
68 end
69
70 for m in 2:M-1
71     # calculate pi and qi for Thomas' algorithm
72     p = zeros(M);
73     q = zeros(M);
74     p[2], q[2] = 0.0, asoln(xs[m], 0.0, ts[n+1]);
75     for l=2:L-1
76         dd = u_temp[m, l] - (2 * μ *
77             (u[m, l+1, n] - 2 * u[m, l, n] + u[m, l-1, n]));
78         denom = aay * p[l] + bby;
79         p[l+1] = -ccy / denom;
80         q[l+1] = (dd - aay * q[l]) / denom;
81     end
82     u[m, L, n+1] = asoln(xs[m], 1.0, ts[n+1]);
83     for l=L-1:-1:2
84         u[m, l, n+1] = p[l+1] * u[m, l+1, n+1] + q[l+1];
85     end
86     u[m, 1, n+1] = asoln(xs[m], 0.0, ts[n+1]);
87 end
88
89 u_exact = zeros(M, L);
90 for m in 1:M, l in 1:L
91     u_exact[m, l] = asoln(xs[m], ys[l], ts[n+1]);
92 end
93
94 if n % output_step == 0
95     println("t=$(ts[n+1]), relative L $\infty$  error: ", norm(u[:, :, n+1] - u_exact,
Inf) / norm(u_exact, Inf));
96     println("t=$(ts[n+1]), relative L2 error: ", norm(u[:, :, n+1] - u_exact,
2) / norm(u_exact, 2));
97     open(w -> begin
98         for m in 1:M, l in 1:L
99             write(w, "$(xs[m]),$(ys[l]),$(u[m, l, n+1]),$(asoln(xs[m], ys[l], ts[n
+1]))\n");
100         end
101     end, dfilename(h, ts[n+1]), "w");
102 end
103 end
104
105 u_exact = zeros(M, L);
106 for m in 1:M, l in 1:L
107     u_exact[m, l] = asoln(xs[m], ys[l], ts[K]);
108 end
109 push!(errs, maximum(map(x -> abs(x), u[:, :, K] - u_exact)));
110 println("t=1.0, relative L $\infty$  error: ", norm(u[:, :, K] - u_exact, Inf) / norm
(u_exact, Inf));
111 println("t=1.0, relative L2 error: ", norm(u[:, :, K] - u_exact, 2) / norm
(u_exact, 2));
112 println();
113
114 open(w -> begin
115     for m in 1:M, l in 1:L
116         write(w, "$(xs[m]),$(ys[l]),$(u[m, l, K]),$(asoln(xs[m], ys[l], ts[K]))
\n");
117     end
118 end, dfilename(h), "w");
119 end
120
121 println(@sprintf("%10s %10s %10s", "h", "max(|e|)", "ratio"));

```

```
122 println(@sprintf("%10.4lf %10.4lf %10s", hs[1], errs[1], "N/A"));
123 println(@sprintf("%10.4lf %10.4lf %10lf", hs[2], errs[2], errs[1]/errs[2]));
124 println(@sprintf("%10.4lf %10.4lf %10lf", hs[3], errs[3], errs[2]/errs[3]));
```