

## MATH 128A, SUMMER 2009: PROGRAMMING ASSIGNMENT 3: SOLUTIONS

- (1) Let's change the notation for Newton's method so it doesn't conflict with the notation for ODEs: Newton's method finds a root of  $g(x)$  using the iteration  $p_{j+1} = p_j - \frac{g(p_j)}{g'(p_j)}$ .

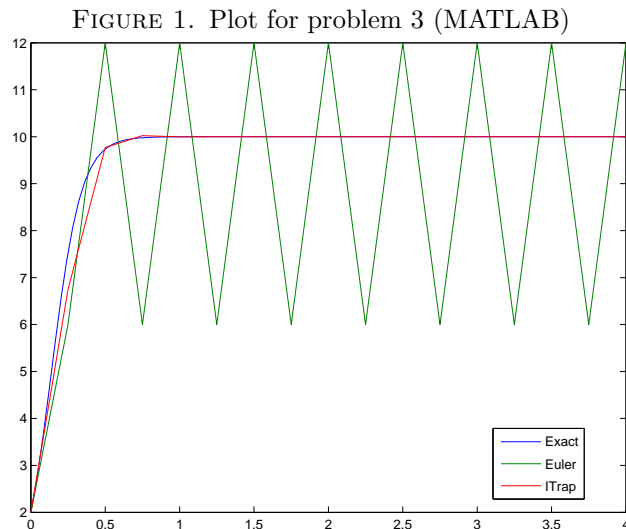
The equation for  $w_{i+1}$  is equivalent to  $g(x) = 0$ , where  $g(x) = (w_i + \frac{h}{2}[f(t_{i+1}, x) + f(t_i, w_i)]) - x$ . To use Newton's method, we need this function's derivative:  $g'(x) = \frac{h}{2} \frac{\partial f}{\partial y}(t_{i+1}, x) - 1$ . Write down:

$$\begin{cases} p_0 = w_i \\ p_{j+1} = p_j - \frac{w_i + \frac{h}{2}[f(t_{i+1}, p_j) + f(t_i, w_i)] - p_j}{\frac{h}{2} \frac{\partial f}{\partial y}(t_{i+1}, p_j) - 1} \end{cases}$$

- (2) In `implicit_trap.m`:

```
function [ti,wi]=implicit_trap(f,dfdy,a,b,y0,N,tol,max_iterations)
h = (b-a)/N;
ti = linspace(a, b, N+1);
wi(1) = y0;
for i=1:N
    % Set up a root-finding problem, g(w(i+1)) = 0.
    g = @(x) -x + wi(i) + h/2 * ( f (ti(i+1),x) + f(ti(i),wi(i)));
    dg = @(x) -1 + h/2 * (dfdy(ti(i+1),x) );
    wi(i+1) = newton(g, dg, wi(i), tol, max_iterations);
end
```

- (3) `f = @(t,y) y.*(10-y); dfdy = @(t,y) 10-2*y; y0 = 2;`  
`tExact = linspace(0, 4);`  
`wExact = 10 ./ (1 + 4*exp(-10*tExact));`  
`[tEuler, wEuler] = euler(f, 0, 4, y0, 4*4);`  
`[tITrap, wITrap] = implicit_trap(f, dfdy, 0, 4, y0, 4*4, 1e-4, 20);`  
`plot(tExact, wExact, tEuler, wEuler, tITrap, wITrap)`  
`legend('Exact', 'Euler', 'ITrap',0)`



(4) Since the values of  $h$  are  $1/2^{i+2}$ ,  $1 \leq i \leq 4$ :

```
for i=1:4
    h(i) = 1/2^(i+2);
    [t, w] = implicit_trap(f, dfdy, 0, 1, y0, 1/h(i), 1e-4, 20);
    err(i) = abs( w(end) - 10 ./ (1 + 4*exp(-10*1)) );
    fprintf('h = 1/%2d: Absolute error = %7.5e\n', 2^(i+2), err(i))
end
```

Output is

```
h = 1/ 8: Absolute error = 1.06856e-03
h = 1/16: Absolute error = 2.98302e-04
h = 1/32: Absolute error = 7.62858e-05
h = 1/64: Absolute error = 1.91750e-05
```

The line “`loglog(h,err,'-o')`” produces figure 2. We need to estimate its slope somehow. Examples:

```
>> format long
>> polyfit(log(h), log(err), 1) % Slope of best-fit line
```

ans =

```
1.936816828682484 -2.784197918366014
```

```
>> ( log(err(4))-log(err(3)) ) / ( log(h(4))-log(h(3)) ) % Slope between 2 points
```

ans =

```
1.992185910508250
```

...or even an “eyeball” estimate from the graph. Any estimate should indicate an order near  $O(h^2)$ .

