

## Math 273 Final Exam

Carry out the *solution* of each problem: show steps of any required calculations; state reasons that justify any conclusions. Mere oracular *answers* will receive no credit.

1. How many floating point operations are required to execute the function `UTriSol` of page 212?

2. Given data vectors  $\mathbf{x}$  and  $\mathbf{y}$  each of length  $m \geq 10$ , how should matrix  $A$  and vector  $\mathbf{b}$  be set up so that the MATLAB command

```
>> a = A \ b
```

computes coefficients in a least squares polynomial  $y = a_1x^2 + a_2x + a_3$  fit to these data? What algorithm does MATLAB execute in response to the backslash command?

3. The function `SineMercEarth` is displayed on page 292. The comment line describes its output as “The sine of the Mercury-Sun-Earth angle at time  $t$ .” Verify or refute this description.

The text goes on to say, “at the time of conjunction, this function is zero.” Is that statement correct?

4. On page 289 a “framework” for Newton’s method is discussed that computes derivatives by finite differences. Replace the line “Choose delta” in the MATLAB fragment there by code to determine a good value of  $\delta_c$ . Be sure to take account of  $|x_c|$ ,  $|f(x_c)|$  and the machine precision.

5. The charge on the capacitor in a series RLC circuit is governed by the stiff second-order differential equation

$$LQ'' + RQ' + \frac{1}{C}Q = E_0 \cos \omega t$$

with  $L = .1$  henry,  $R = 8 \times 10^3$  ohms,  $C = 0.25 \times 10^{-6}$  farad,  $E_0 = 110$  volts, and  $\omega = 120\pi$ .

The MATLAB ODE solvers accept differential equations in the form of first-order systems. Convert the second-order differential equation into a first order system, and fill in the code for it below.

```
% -----
```

```
function dydt = f(t,y,p1,p2)
dydt = < Insert a function of t and/or y, p1, and p2 here. >
```

```
% -----
```

Since the problem is stiff, the efficiency and accuracy of the stiff solver can be enhanced by providing the analytical Jacobian. Fortunately this is easy to do, since the differential equation is linear. Fill it in below.

```
% -----
```

```
function dfdy = jacobian(t,y,p1,p2)
dfdy = < Insert Jacobian matrix here. >;
```

```
% -----
```