National Taiwan University
Department of Civil Engineering

Finite Element Method
Instructors: C.-S. CHEN

Lab Assignment 2, 03/21/2019, 1800 -- 2000

Due 2000

Lab Grading Policy: Attendance 20%, Score 80%

You are expected to complete the basic part during the Lab. In case you have difficulty in finishing the basic part on time, you should upload them before 2100 on Saturday and a penalty of 20% discount will be applied on your score. You are encouraged to complete the bonus part (no penalty applied). Basic and/or bonus parts should be submitted by **2100 on Saturday and no late submission is permitted**. We will in general post the reference solutions by **Sunday**.

---

1. (40%) Implement a home-made square root function `sqrtn` using a simple Newton iteration. Given $a > 0$, a simple Newton iteration to find $\sqrt{a}$ is:

$$x_{k+1} = \frac{1}{2}(x_k + \frac{a}{x_k}) \qquad x_0 = a$$

For example, $\sqrt{\ }$ for the first four iterations and its relative change $\frac{|x_{k+1} - x_k|}{|x_{k+1}|}$ are:

```
k           x_k                rel. change
1:  1.5000000000000000e+000  3.33e-001
2:  1.4166666666666665e+000  5.88e-002
3:  1.4142156862745097e+000  1.73e-003
4:  1.4142135623746899e+000  1.50e-006
```

Notice that the above output is accomplished by `fprintf`. Implement this home-made square root function with the following function prototype:

```
function [x,iter] = sqrtn(a, tol)
```

in which `tol` is the tolerance for convergence ($\frac{|x_{k+1} - x_k|}{|x_{k+1}|} < \text{tol}$) (the default is `eps`) and `iter` is the number of iteration. In addition, (1) implement your function using `nargin` to determine whether you need to set the default tolerance, and (2) implement the corresponding comments for `help`.

Below are a few sample runs:

```
>>
```

```
 sqrtn   Square root of a scalar by Newton's method.
        X = sqrtn(A,TOL) computes the square root of the scalar
        A by Newton's method (also known as Heron's method).
        A is assumed to be >= 0.
        TOL is a convergence tolerance (default EPS).
        [X,ITER] = sqrtn(A,TOL) returns also the number of
        iterations ITER for convergence.

>> [      ] =      (2, 0.01)
 k           x_k          rel. change
 1:  1.5000000000000000e+00   3.33e-01
 2:  1.4166666666666665e+00   5.88e-02
 3:  1.4142156862745097e+00   1.73e-03

x =

   1.4142


iter =

    3

>> [      ] =      (2)
 k           x_k          rel. change
 1:  1.5000000000000000e+00   3.33e-01
 2:  1.4166666666666665e+00   5.88e-02
 3:  1.4142156862745097e+00   1.73e-03
 4:  1.4142135623746899e+00   1.50e-06
 5:  1.4142135623730949e+00   1.13e-12
 6:  1.4142135623730949e+00   0.00e+00

x =

   1.4142


iter =

    6
```

2. (40%) Let `x` and `y` be column vectors describing the vertices of a polygon, given in order. Write functions `mypolyperim(x,y)` and `mypolyarea(x,y)` that compute the perimeter and area of the polygon. For the area, use a formula below:

$$A = \frac{1}{2}\left|\sum_{k=1}^{n} x_k y_{k+1} - x_{k+1} y_k\right|$$

in which n is the number of polygon vertices, and by definition $x_{n+1} = x_1$ and $y_{n+1} = y_1$.

(a) Test your functions on a square and an equilateral triangle.

(b) Compare your area calculation results with MATLAB built-in function `polyarea(x,y)`.

Below are sample outputs:

```
Test #1: Square
* Coordinates:
 Node 1 : (+0.0000,+0.0000)
 Node 2 : (+1.0000,+0.0000)
 Node 3 : (+1.0000,+1.0000)
 Node 4 : (+0.0000,+1.0000)
* Perimeter:
 mypolyperim(x,y)  =  4.000000
 Exact perimeter   =  4.000000
 Error (%)         =  0.000000
* Area:
 mypolyarea(x,y)   =  1.000000
 polyarea(x,y)     =  1.000000
 Error (%)         =  0.000000
Test #2: Equilateral Triangle
* Coordinates:
 Node 1 : (-0.5000,+0.0000)
 Node 2 : (+0.5000,+0.0000)
 Node 3 : (+0.0000,+0.8660)
* Perimeter:
 mypolyperim(x,y)  =  3.000000
 Exact perimeter   =  3.000000
 Error (%)         =  0.000000
* Area:
 mypolyarea(x,y)   =  0.433013
 polyarea(x,y)     =  0.433013
 Error (%)         =  0.000000
```

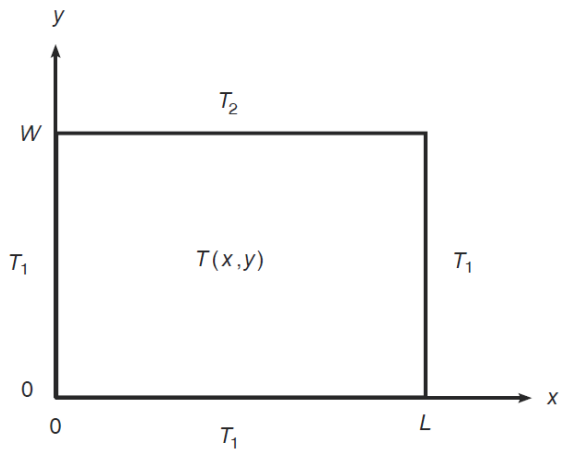
3. **(Bonus 20%)** The following equation describes the temperature distribution in a rectangular metal plate. The temperature is held constant at T_1 on three sides and at T_2 on the fourth side as shown below. The temperature $T(x\ y$ as a function of the xy coordinates shown is given by:

$$T(x,y) = (T_2 - T_1)w(x,y) + T_1$$

in which

$$w(x,y) = \frac{2}{\pi}\sum_{n\ \text{odd}}^{\infty}\frac{2}{n}\sin(\frac{n\pi x}{L})\frac{\sinh(n\pi y/L)}{\sinh(n\pi W/L)}$$

In our case, we will let $T_1 = 70^\circ F$, $T_2 = 200^\circ F$ and $W = L = 2$ ft..



Write a MATLAB function `temperatureDist.m` to obtain the solution of a given coordinate and let user specify the desirable accuracy. For example, if we want the calculation to be accurate within 1 percent, the addition of the next term in the series should produce a change in T of less than 1 percent. Use $x = y = 1$ to test your function. A sample run is given below:

```
>> T = temperatureDist(1,1,0.01)

T =

  102.4875
```