COMP5930M-Scientific Computation

Coursework 1

Daolin Sheng ml192ds

1. A floating sphere

(a). Nonlinear function and MATLAB implementation

- The weight of water $W_w = \rho_w V_{cap} = \pi H^2(a - \frac{H}{3})$ and the weight of the sphere $W_s = \frac{4}{3}\pi\rho_s a^3$.
- Because of $W_w = W_s$, so the nonlinear equation $F_{a,\rho_s}(H) = W_w - W_s = 0$. the nonlinear equation is down for given values of a and ρ_s:

$$F_{a,\rho_s}(H) = \pi H^2(a - \frac{H}{3}) - \frac{4}{3}\pi\rho_s a^3 = 0$$

- The implementation of MATLAB:

```
function f_h = fun_sphere(H, a, rhos)
    % The nonlinear equations Fa,phos (H) = 0
    % the weight of the water: W_w = pi*H.^2*(a - H/3)
    % the weight of the sphere: W_s = 4/3*pi*rhos*a.^3
    %
    % function f_h = fun_sphere(H, a, rhos)
    %
    % Input:
    %     H - a depth below the water surface
    %     a - the radius of sphere, constant value
    %     rhos- the density of sphere, constant value
    %
    % Output:
    %    f_h - final function value
    %

    f_h = pi*H.^2*(a - H/3) - 4/3*pi*rhos*a.^3;

end
```

(b). The initial bracket plus justification for its validity

- the initial bracket: $[0+\delta, 3a-\delta], \delta = 10^{-6}$

> Because the H_L and H_R must meet the following conditions:
>
> - (1). $0 < H < 3a$, due to the $W_w > 0$
> - (2). $F(x_L)F(x_R) \leq 0$
> - (3). The F(H) can be get a maximum at the point $2a$, So, the suitable value can be $[0+\delta, 2a], \delta = 10^{-6}$ OR $[2a, 3a-\delta], \delta = 10^{-6}$ must be find a H to fit the $F(H)$ = 0.

(c). Two correct solutions plus the number of iterations

- In given values $a = 1$ and $p_s = 0.45$,
- In Bisection method, the initial bracket $[10^{-6}, 3 - 10^{-6}]$, the solution is : $H = 0.9332$ and the number of iteration is **22.**

- In Newton Method, the init guess $x_0 = 1.0$, the solution is : $H = 0.9332$ and the number of iteration is **3.**
- As the following figure:

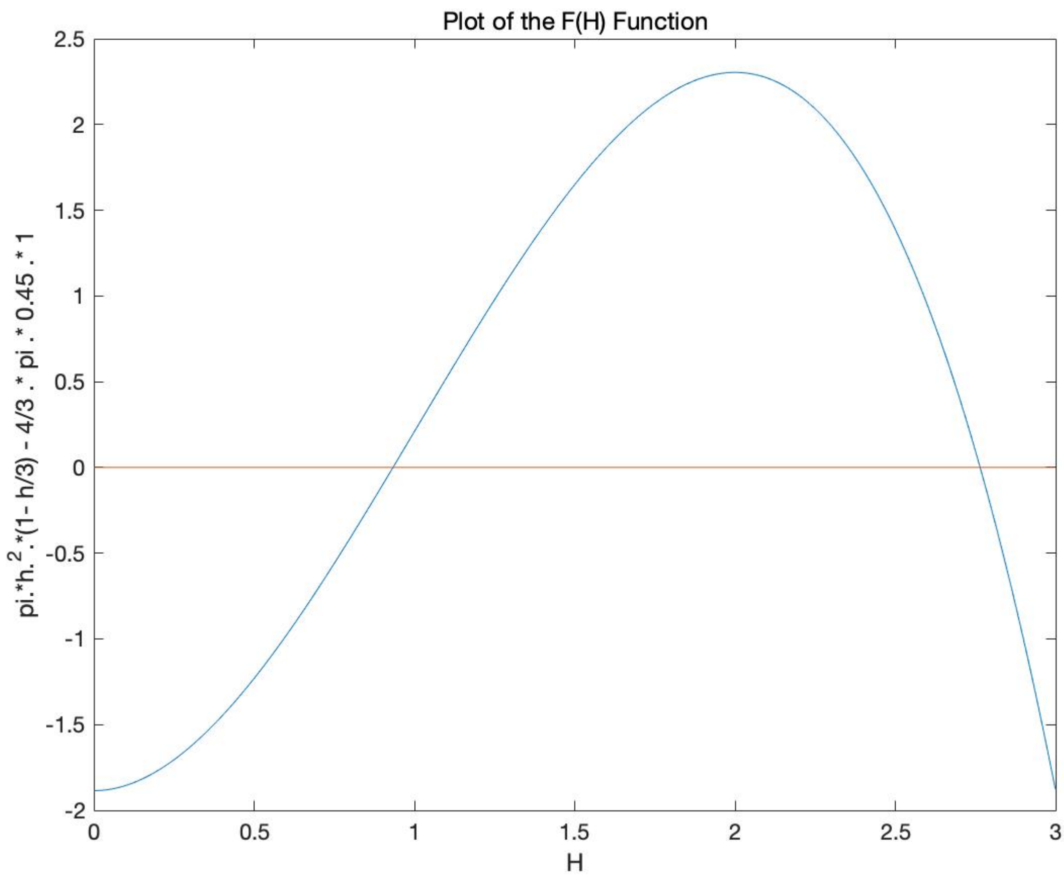
```
>> bisection(@fun_sphere, 1e-6, 2-1e-6, 1e-6, 100)
   i   x_i          |F(x_i)|
   0   1.00000000   2.09e-01
   1   0.50000050   1.23e+00
   2   0.75000025   5.60e-01
   3   0.87500013   1.81e-01
   4   0.93750006   1.33e-02
   5   0.90625009   8.42e-02
   6   0.92187508   3.55e-02
   7   0.92968757   1.11e-02
   8   0.93359382   1.12e-03
   9   0.93164069   4.98e-03
  10   0.93261725   1.93e-03
  11   0.93310554   4.02e-04
  12   0.93334968   3.61e-04
  13   0.93322761   2.04e-05
  14   0.93328864   1.70e-04
  15   0.93325812   7.51e-05
  16   0.93324286   2.73e-05
  17   0.93323524   3.47e-06
  18   0.93323142   8.46e-06
  19   0.93323333   2.50e-06
  20   0.93323428   4.85e-07
  21   0.93323428   4.85e-07

ans =

  列 1 至 8

    1.0000    0.5000    0.7500    0.8750    0.9375    0.9063    0.9219    0.9297

  列 9 至 16

    0.9336    0.9316    0.9326    0.9331    0.9333    0.9332    0.9333    0.9333

  列 17 至 21

    0.9332    0.9332    0.9332    0.9332    0.9332

fx >>
```

```
>> newtonSys(@fun_sphere, @dfun_sphere, 1.0, 1e-6, 100)
   k   |F(x_k)|
   0      0.209
   1    0.00031
   2   2.06e-09
Converged

ans =

    0.9332

fx >>
```

(d). number of solutions and corresponding initial guesses

- By changing the initial guess x_0, can be get **2** solutions, the first one is:

- $H = 0.9332$, the second one is $H = 2.7645$.



- if the initial guess $x_0 = 0.5$, the solution is : **0.9332.**

```
>> newtonSys(@fun_sphere, @dfun_sphere, 0.5, 1e-6, 100)
   k    |F(x_k)|
   0        1.23
   1       0.279
   2    0.000184
   3    7.28e-10
Converged

ans =

    0.9332

>>
```

- if the initial guess $x_0 = 2.5$, the solution is : **2.7645.**

```
>> newtonSys(@fun_sphere, @dfun_sphere, 2.5, 1e-6, 100)
   k    |F(x_k)|
   0        1.39
   1       0.635
   2      0.0395
   3    0.000192
   4    4.66e-09
Converged

ans =

    2.7645

>> |
```

2. Chemical engineering

(a). Nonlinear system for the steady-state problem and MATLAB code

For the case **n=5** reactors, the nonlinear equation system $F(U) = 0$.

```
function f = fun_chemical(a, V, G, k, a0)

    % The nonlinear equation system F(U)
    % function f = fun_chemical(a, V, G, k, a0)
    % Input:
    %       a - current solution
    %       V - constants
    %       G - constants
    %       k - constants
    %      a0 - constants
    % Output:
    %      f = final function value
    f = [(k*V/G)*a(1).^2 - a0 + a(1);
         (k*V/G)*a(2).^2 - a(1) + a(2);
         (k*V/G)*a(3).^2 - a(2) + a(3);
         (k*V/G)*a(4).^2 - a(3) + a(4);
         (k*V/G)*a(5).^2 - a(4) + a(5)
         ];
end
```

(b). Jacobian matrix for the steady-state problem and MATLAB code

the Jacobian matrix

```
function jf = Jfun_chemical(a, V, G, k, a0)
    % The nonlinear equation system F(U)
    % function f = fun_chemical(a, V, G, k, a0)
    % Input:
    %       a - current solution
    %       V - constants
    %       G - constants
    %       k - constants
    %      a0 - constants
    % Output:
    %      jf = Jacobian matrix
   jf = [
          (k*V/G)*2*a(1) + 1, 0, 0, 0, 0;
         -1, (k*V/G)*2*a(2) + 1, 0, 0, 0;
         0, -1, (k*V/G)*2*a(3) + 1, 0, 0;
         0, 0, -1, (k*V/G)*2*a(4) + 1, 0;
         0, 0,  0, -1, (k*V/G)*2*a(5) + 1
        ];
end
```

(c). Numerical solution and values of |F|

- In case the **n =5** with the following the parameters: **V = 1.0, G = 35.0, k=0.6, a0= 6.0.**
- The values of $|F|$ at each iterations:

  iterations: |f(x)|

  0 6.000000

  1 1.379973

  2 0.078314

  3 0.000231

  4 0.000000

  5 0.000000

  the solutions:

  5.4844

  5.0476

  4.6732

  4.3490

  4.0656

(d). Discreised time-dependent equations

- the time-dependent concentrations $a_i(t)$.
  (1). $\frac{da_i}{dt} = -\beta a_i^2 + a_{i-1} - a_i$, for $t > 0$; and $a_i = a(t_i)$
- Implicit Euler approximation for (2). $\frac{da_i}{dt} = f(u)$
- and the backward Euler(implicit) method is: (3). $\frac{a_{i+1}-a_i}{\Delta t} = f(u_{i+1})$
- So we can get this formual:
- $F_i(U) = \frac{a_{i+1}-a_i}{\Delta t} + \beta a_{i+1}^2 + a_{i+1} - a_i = 0$ given an initial $a_i(0) = a_i^0$ for i = 1, ..., n.

(e). Jacobian matrix for the time-dependent problem, behaviour for limit values of times step

- the Jacpbian matix is :

$$\begin{bmatrix} \dfrac{\partial F_i}{\partial a_i} \\ \dfrac{\partial F_i}{\partial a_{i+1}} \end{bmatrix} = \begin{bmatrix} -1 - \dfrac{1}{\Delta t} \\ 1 + \dfrac{1}{\Delta t} + 2\beta a_{i+1} \end{bmatrix}$$

- when (Δt -> 0), the Jacobian matrix is trend to $[-1; \infty]$, this reactor is unsteady-state.
- when (Δt -> ∞), and this reactor is steady-state.

3. Control of a robot arm

(a) System of equations

- The system of nonlinear equations in the form **F(x)** = **0.**
- Base on the equations of the location of the free end (loc_x, loc_y) and $(x_1, x_2) = (\theta, \phi)$:

$$loc_x = cos(\theta) + cos(\phi)$$

$$loc_y = sin(\theta) + sin(\phi)$$

- So, **x** is the vector $\{x_1, x_2\}$, **F** is a set $\{F_1$(**x**), F_2(**x**)$\}$ nonlinear equations:

$$F_1(x_1, x_2) = cos(x_1) + cos(x_2) - loc_x$$

$$F_2(x_1, x_2) = sin(x_1) + sin(x_2) - loc_y$$

- Find (x_1^*, x_2^*) such that $F_1(x_1^*, x_2^*) = 0$ and $F_2(x_1^*, x_2^*) = 0$.

$$cos(x_1) + cos(x_2) - loc_x = 0$$

$$sin(x_1) + sin(x_2) - loc_y = 0$$

(b) Implement in MATLAB code

the matlab filename: **fun_arm.m**

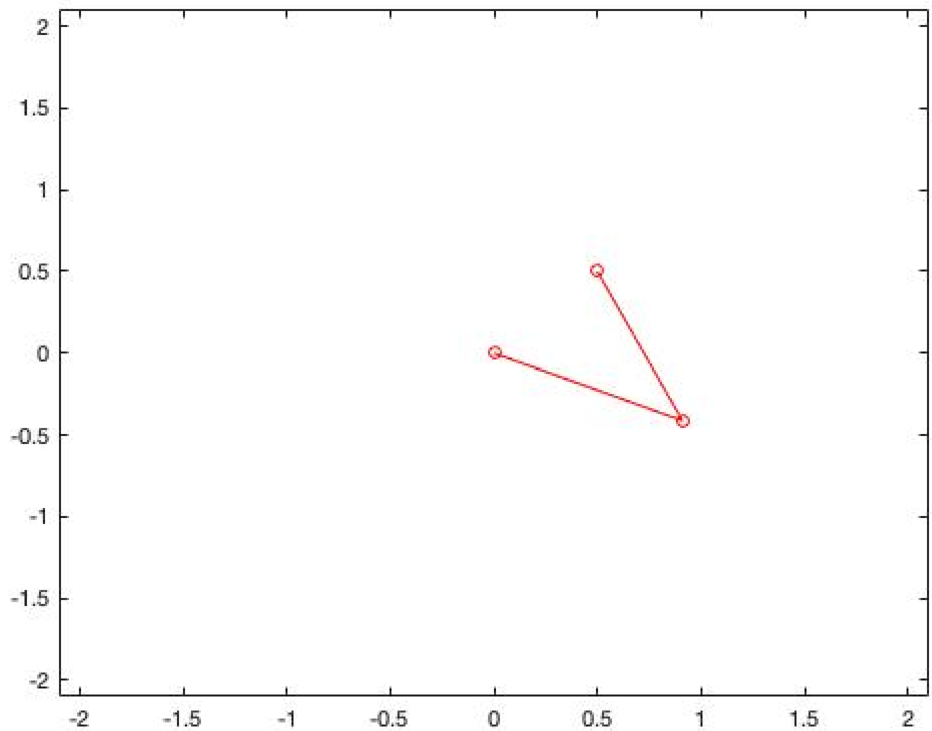
```
function f = fun_arm(x, locx, locy)
    % Systems of nonlinear equations for control of a robot arm
    % system of 2 nonlinear equations
    % function f_x = fun_arm(x, locx, locy)
    %
    % Input: x - current solution
    %        locx - current location x
    %        locy - current location y
    %
    % Output: f - final function value

    f = [cos(x(1)) + cos(x(2)) - locx;
         sin(x(1)) + sin(x(2)) - locy];
end
```

The Jacobian function: **Jfun_arm(x)**:

```
function jf = Jfun_arm(x)
  % Analytical Jacobian
  % function jf =  trueJacobian(x)
  %
  % Input:
  %        x - current solution
  %
  %
  % Output: jf - Jacobian matrix

  jf = [-sin(x(1)), -sin(x(2)); cos(x(1)), cos(x(2))];

end
```

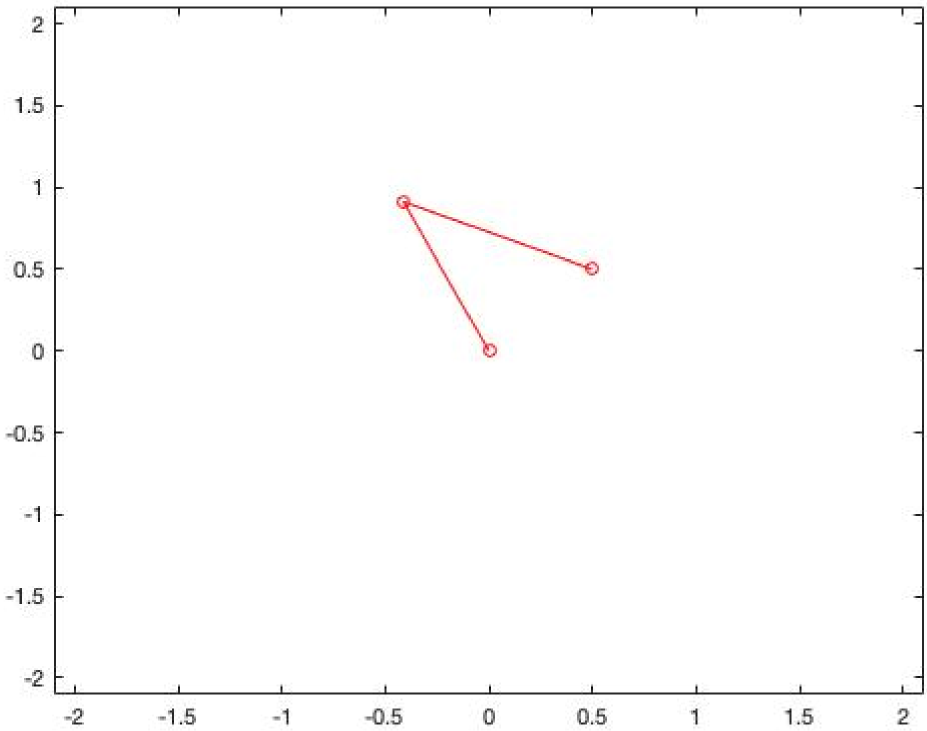
(c) two correct solutions, figures

- **Case i**, the initial x_0 for Newton method, when x_0 at location $(x_1, x_2) = (-1, 1)$
  the solution is: $(\theta, \phi) = (x_1, x_2) = (-0.4240, 1.9948)$
  the figure:



- **Case ii**, the initial x_0 for Newton method, when x_0 at location $(x_1, x_2) = (2, 0)$
  the solution is: $(\theta, \phi) = (x_1, x_2) = (1.9948, -0.4240)$

the figure:



(d) Implement in MATLAB code about the path and the tracing function, and the table of angles

1. the traceFn is the function defining the path. the code: filename: **traceFn.m**

```
function [locx, locy] = traceFn(t)
    % traceFn is the function defining path
    % of the free end of the arm.
    % trace path defined by the following fucniton
    % (x,y)=(t, 0.1+sin(2t-0.5)), and (-1<=t<= 1)
    %
    % function [locx, locy] = traceFn(t)
    % Input: t - param variables,from t=-1 to t=1
    %
    % Output: locx - the location point x
    %         locy - the location point y

    locx = t;
    locy = 0.1 + sin(2*t - 0.5);

```

```
end
```

2. The matlab filename: **“traceArm.m”**

```
function t_out = traceArm(traceFn,nSteps, x0)

    % Trace path of the free end of arm at initial x0
    % function t_out = traceArm(traceFn,nSteps, x0)
    % Input:
    %    traceFn -to create the location (x,y)
    %    nSteps -the number of steps for t splitting the t
    %    x0     -the init guess value for Newton method
    % Output:
    %    t_out  - a table to recording the angles
    %    theta and phi at each step.
    step = 2 / (nSteps -1);
    i = 1;

    step_list = [];
    t = [];
    x_list = [];
    y_list = [];
    theta_list = [];
    phi_list = [];
    for st = -1:step:1
        [locx, locy] = feval(traceFn, st)
        [xx, f] = newtonSys2(@fun_arm, @Jfun_arm, x0, 1e-10, 100,locx,locy);
        t = [t st];
        step_list = [step_list i];
        x_list = [x_list locx];
        y_list = [y_list locy];
        theta_list = [theta_list xx(1,:)];
        phi_list = [phi_list xx(2,:)];
        i = i + 1;
    end
    nSteps = step_list';
    nT = t';
    nLocx  = x_list';
```

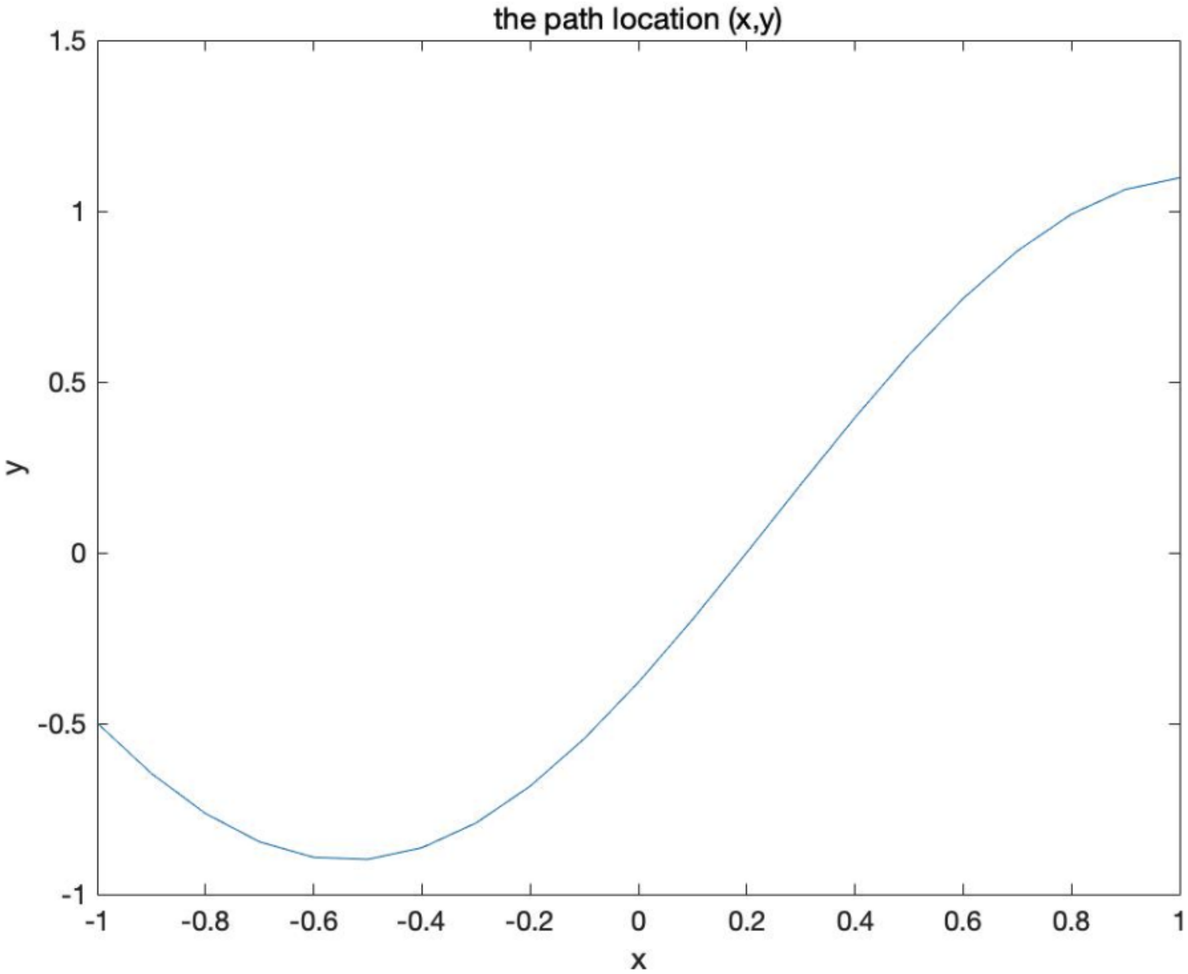
```
    nLocy  = y_list';
    Theta  = theta_list';
    Phi    = phi_list';
    t_out = table(nSteps,nT, nLocx, nLocy,Theta,Phi);

end
```

3. In case the **x0: (0.5, 0.5)**, the θ and ϕ at each step, following down:

21x6 table

	1 nSteps	2 nT	3 nLocx	4 nLocy	5 Theta	6 Phi	7
1	1	-1	-1	-0.4985	-1.7012	2.6260	
2	2	-0.9000	-0.9000	-0.6457	-1.5354	2.7801	
3	3	-0.8000	-0.8000	-0.7632	48.8708	53.1839	
4	4	-0.7000	-0.7000	-0.8463	-3.2513	-1.2724	
5	5	-0.6000	-0.6000	-0.8917	-1.1596	3.1166	
6	6	-0.5000	-0.5000	-0.8975	3.1728	5.2354	
7	7	-0.4000	-0.4000	-0.8636	-7.2129	-3.0794	
8	8	-0.3000	-0.3000	-0.7912	-0.7993	3.2160	
9	9	-0.2000	-0.2000	-0.6833	-0.6487	3.2208	
10	10	-0.1000	-0.1000	-0.5442	-0.4620	3.2402	
11	11	0	0	-0.3794	3.3325	6.0923	
12	12	0.1000	0.1000	-0.1955	28.8571	31.7787	
13	13	0.2000	0.2000	1.6658e-04	-1.4698	1.4715	
14	14	0.3000	0.3000	0.1998	-0.8020	1.9772	
15	15	0.4000	0.4000	0.3955	-0.5059	2.0655	
16	16	0.5000	0.5000	0.5794	-0.3193	2.0370	
17	17	0.6000	0.6000	0.7442	-0.1802	1.9647	
18	18	0.7000	0.7000	0.8833	-0.0715	1.8728	
19	19	0.8000	0.8000	0.9912	0.0114	1.7721	
20	20	0.9000	0.9000	1.0636	0.0684	1.6686	
21	21	1	1	1.0975	0.0977	1.5660	
22							

The path of free end of arm.



The newtonSys2 function

```
function [ x,f ] = newtonSys2( fnon, fjac, x0, tol, maxIt,locx,locy)

% Basic Newton algorithm for systems of nonlinear equations
% function [ x,f ] = newtonSys( fnon, fjac, x0, tol, maxit )
% Input: fnon - function handle for nonlinear system
%        fjac - function handle for Jacobian matrix
%        x0 - initial state (column vector)
%        tol - convergence tolerance
%        maxIt - maximum allowed number of iterations
% Output: x - final point
%         f - final function value

fprintf(' x      |f(x)|\n')
```

```matlab

n = length(x0);

x = x0;                 % initial point
f = feval(fnon,x,locx, locy);      % initial function values
normf = norm(f);
it = 0;
fprintf(' %d %12.6f\n',it,norm(f));

while (normf>tol) && (it<maxIt)

  J = feval(fjac, x);  % build Jacobian
  delta = -J\f;        % solve linear system

  xkp = x;
  xk  = x + delta;

  x = x + delta;       % update x
  f = feval(fnon,x, locx, locy);   % new function values
  normf = norm(f);

  it = it + 1;
  % Print the new estimate and function value.
  fprintf(' %d %12.6f\n',it,normf)

  %plot([xkp(1) xk(1)], [xkp(2) xk(2)], 'ok-', 'LineWidth', 2)
  %pause
end

if( it==maxIt)
    fprintf(' WARNING: Not converged\n')
else
    fprintf(' SUCCESS: Converged\n')
end
```