Newton's Method for Multidimension System

Table of Contents

Problem.

Suppose one wants to find the points on the ellipsoid $x^2/25 + y^2/16 + z^2/9 = 1$ that are closest to and farthest from the point $(5, 4, 3)$. The method of Lagrange multipliers implies that any such point satisfies

$$\begin{cases} x - 5 &= \frac{\lambda x}{25}, \\ y - 4 &= \frac{\lambda y}{16}, \\ z - 3 &= \frac{\lambda z}{9}, \\ 1 &= \frac{1}{25}x^2 + \frac{1}{16}y^2 + \frac{1}{9}z^2 \end{cases}$$

for an unknown value of λ.

(a) Write out this system in the form $\mathbf{f}(\mathbf{u}) = \mathbf{0}$.

(b) Write out the Jacobian matrix of this system.

(c) Use `newtonsys` from class with different initial guesses to find the two roots of this system. Which is the closest point to $(5, 4, 3)$ and which is the farthest?

Solution.

(a)

Moving all terms to one side and simplifying, we have

$$
\begin{aligned}
\left(1-\frac{\lambda}{25}\right)x-5 &= 0\\
\left(1-\frac{\lambda}{16}\right)y-4 &= 0\\
\left(1-\frac{\lambda}{9}\right)z-3 &= 0\\
\frac{1}{25}x^2+\frac{1}{16}y^2+\frac{1}{9}z^2-1 &= 0
\end{aligned}
$$

This is good enough. But for the last part, it is advantageous to write in the form $\mathbf{f}(\mathbf{u})=\mathbf{0}$ as suggested by the problem. Let $\mathbf{u}=(u_1,u_2,u_3,u_4)^{\mathrm{T}}=(x,y,z,\lambda)^{\mathrm{T}}$, the vector consisting of all four unknowns. Then, abstractly, we may view/write the system of these four equations as a single vector equation $\mathbf{f}(\mathbf{u})=\mathbf{0}$,

$$
\mathbf{f}(\mathbf{u})=\begin{bmatrix} f_1(u_1,u_2,u_3,u_4)\\ f_2(u_1,u_2,u_3,u_4)\\ f_3(u_1,u_2,u_3,u_4)\\ f_4(u_1,u_2,u_3,u_4)\end{bmatrix}=\begin{bmatrix} (1-u_4/25)u_1-5\\ (1-u_4/16)u_2-4\\ (1-u_4/9)u_3-3\\ u_1^2/25+u_2^2/16+u_3^2/9-1\end{bmatrix}
$$

simply by replacing x,y,z,λ by u_1,u_2,u_3,u_4, respectively.

(b)

In terms of the original variables x,y,z,λ, we can express the Jacobian matrix as

$$
\mathbf{J}(x,y,z,\lambda)=\begin{bmatrix} 1-\lambda/25 & 0 & 0 & -x/25\\ 0 & 1-\lambda/16 & 0 & -y/16\\ 0 & 0 & 1-\lambda/9 & -z/9\\ 2x/25 & 2y/16 & 2z/9 & 0\end{bmatrix}
$$

which can be rewritten in terms of $\mathbf{u}=(u_1,u_2,u_3,u_4)^{\mathrm{T}}=(x,y,z,\lambda)^{\mathrm{T}}$ as

$$
\mathbf{J}(\mathbf{u})=\begin{bmatrix} 1-u_4/25 & 0 & 0 & -u_1/25\\ 0 & 1-u_4/16 & 0 & -u_2/16\\ 0 & 0 & 1-u_4/9 & -u_3/9\\ 2u_1/25 & 2u_2/16 & 2u_3/9 & 0\end{bmatrix}.
$$

The latter will be useful in the next part.

(c)

In order to use `newtonsys` to find roots of $\mathbf{f}$, we first need to write a function m-file calculating both $\mathbf{f}$ and $\mathbf{J}$. Inspired by the example in the live script accompanying Lecture 23, we write

```
function [f,J] = nlsystem(x)
    f = [ (1-u(4)/25)*u(1) - 5;
          (1-u(4)/16)*u(2) - 4;
          (1-u(4)/9)*u(3) - 3;
          u(1)^2/25 + u(2)^2/16 + u(3)^2/9 - 1];
    J = [ 1-u(4)/25, 0, 0, -u(1)/25;
          0, 1-u(4)/16, 0, -u(2)/16;
          0, 0, 1-u(4)/9,  -u(3)/9;
          2*u(1)/25, 2*u(2)/16, 2*u(3)/9, 0];
end
```

(This function is included at the end of this live script.)

Then we are able to use `newtonsys` as follows.

```
format short
x1 = newtonsys(@nlsystem, [1 1 1 1])
```

```
x1 = 4×7
    1.0000    4.6945    3.5843    3.4311    3.4241    3.4241    3.4241
    1.0000    3.4446    2.4818    2.3311    2.3268    2.3268    2.3268
    1.0000    1.8336    1.4283    1.3186    1.3167    1.3167    1.3167
    1.0000  -11.3310  -10.2192  -11.3800  -11.5053  -11.5056  -11.5056
```

Note that the name of the function must be passed with at-sign in its front since it is defined as an m-file. (If you defined a function in-line as an anonymous function, the at-sign is unnecessary.)

To find another solution, use a different initial guess.

```
x2 = newtonsys(@nlsystem, [-1 -1 -1 1])
```

```
x2 = 4×10
   -1.0000    2.1306   -0.8437  -11.7432   -6.6115   -4.7672   -4.4169   -4.4038 ···
   -1.0000   -0.6577   -1.5221   -3.5260   -2.5298   -1.8132   -1.7221   -1.7119
   -1.0000   -5.8583   -3.8719   -1.5987   -0.7062   -0.6950   -0.6045   -0.6084
    1.0000   74.8662   35.9427   31.7917   38.6124   47.7038   52.8890   53.3832
```

Check. Are the solutions really on the ellipsoid?

```
eq_ellips = @(u) u(1)^2/25 + u(2)^2/16 + u(3)^2/9 - 1;
p1 = x1(1:3,end);
p2 = x2(1:3,end);
format short e
[eq_ellips(p1), eq_ellips(p2)]'
```

```
ans = 2×1
            0
   2.2204e-16
```

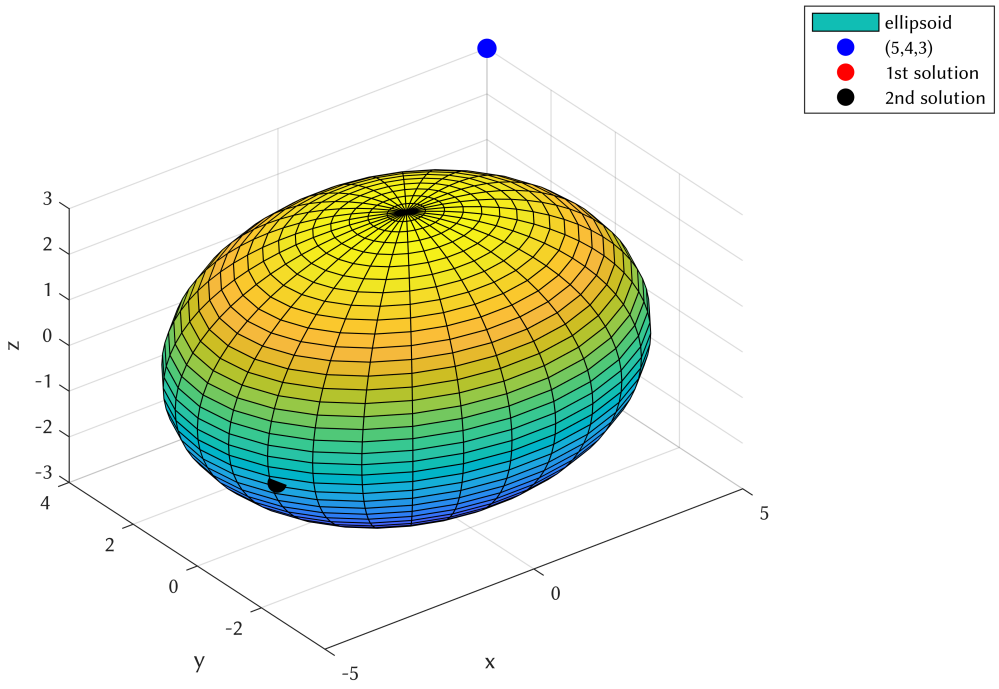
Yes, they are!

Closest or Farthest?

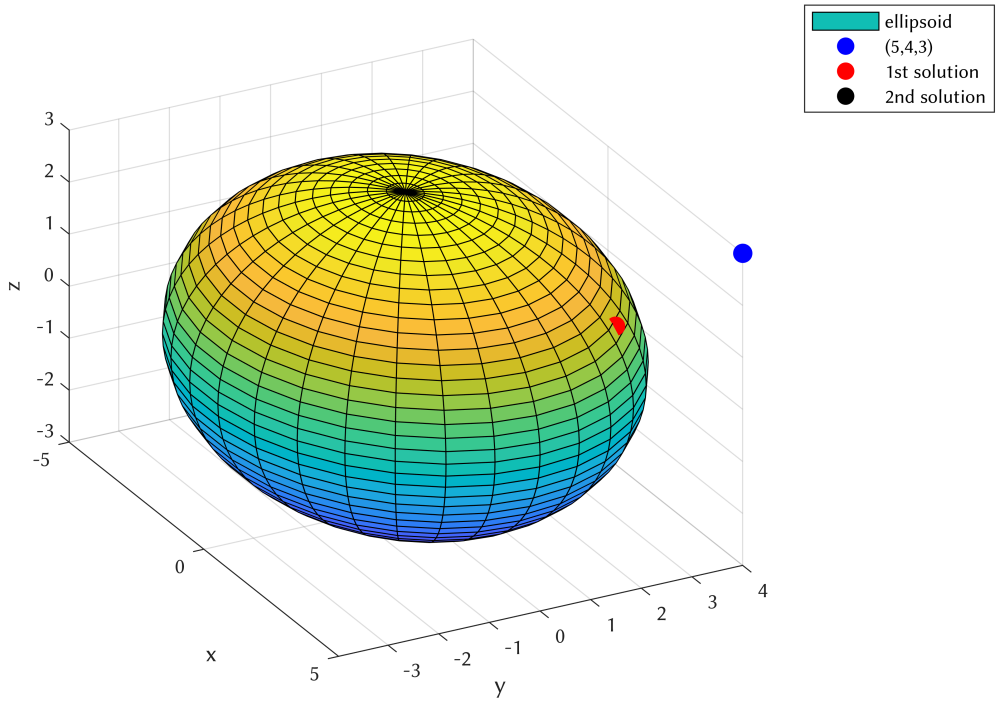
One of the two solutions have all positive components while the other has all negative components. The former should be closest while the latter farthest, because the given point $(5, 4, 3)$ lies in the first octant.

Run the following script and rotate the generated 3-D figure around to stop the closest and the farthest points.

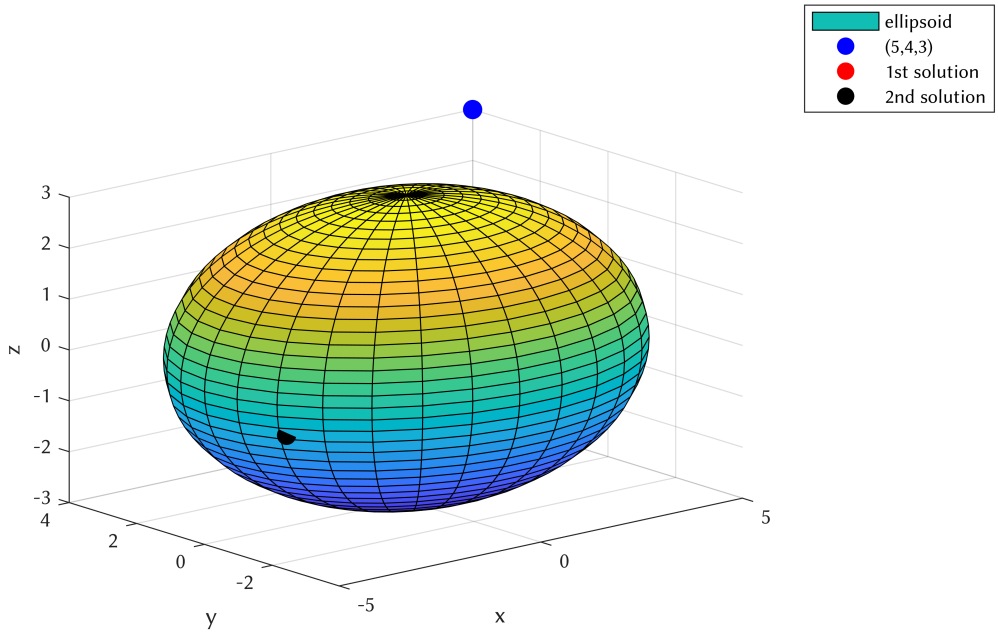
```
nr_th = 31;
nr_ph = 41;
th = linspace(0, 2*pi, nr_th);
ph = linspace(0, pi, nr_ph);
[TH,PH] = meshgrid(th, ph);
a = 5; b = 4; c = 3;
x = a*sin(PH).*cos(TH);
y = b*sin(PH).*sin(TH);
z = c*cos(PH);
clf
surf(x,y,z), hold on, axis equal
xlabel('x'), ylabel('y'), zlabel('z')
plot3(5,4,3, 'b.', 'MarkerSize', 30)
plot3(p1(1), p1(2), p1(3), 'r.', 'MarkerSize', 30)
plot3(p2(1), p2(2), p2(3), 'k.', 'MarkerSize', 30)
legend('ellipsoid', '(5,4,3)', '1st solution', '2nd solution')
```




```
view([62 25])    % better viewing angle for the first solution
```




```
view([-40 15])  % better viewing angle for the second solution
```



Functions.

Newton Iteration for Systems

```
function x = newtonsys(f,x1)
% NEWTONSYS   Newton's method for a system of equations.
% Input:
%   f        function that computes residual and Jacobian matrix
%   x1       initial root approximation (n-vector)
% Output
%   x        array of approximations (one per column, last is best)

% Operating parameters.
    funtol = 1000*eps;  xtol = 1000*eps;  maxiter = 40;

    x = x1(:);
    [y,J] = f(x1);
    dx = Inf;
    k = 1;

    while (norm(dx) > xtol) && (norm(y) > funtol) && (k < maxiter)
        dx = -(J\y);    % Newton step
        x(:,k+1) = x(:,k) + dx;

        k = k+1;
        [y,J] = f(x(:,k));
    end

    if k==maxiter, warning('Maximum number of iterations reached.'), end
end
```

Residual and Jacobian function

```
function [f,J] = nlsystem(u)
    f = [ (1-u(4)/25)*u(1) - 5;
          (1-u(4)/16)*u(2) - 4;
          (1-u(4)/9)*u(3) - 3;
          u(1)^2/25 + u(2)^2/16 + u(3)^2/9 - 1];
    J = [ 1-u(4)/25, 0, 0, -u(1)/25;
          0, 1-u(4)/16, 0, -u(2)/16;
          0, 0, 1-u(4)/9,  -u(3)/9;
          2*u(1)/25, 2*u(2)/16, 2*u(3)/9, 0];
end
```