# Recap Newton's Method.

**Big idea:** To find a root of a nonlinear function, find a series of root estimates by solving simpler problems.

linear approx. of the given function.

**1-D:** $f(x+h) \approx f(x) + f'(x)\,h$

center — small pert.

**n-D:** $\vec{f}(\vec{x}+\vec{h}) \approx \vec{f}(\vec{x}) + J(\vec{x})\,\vec{h}$

Jacobian matrix

$\vec{f}(\vec{x})$: $n \times 1$

$J(\vec{x})$: $n \times n$

$\vec{h}$: $n \times 1$

$J(\vec{x})\,\vec{h}$: $n \times 1$

# Newton Iteration Scheme

We find a root of the linear approx. of $\vec{f}$ at $\vec{x}_k$ ; call it $\vec{x}_{k+1}$.

$$\vec{0} = \vec{f}(\vec{x}_k) + J(\vec{x}_k)(\vec{x} - \vec{x}_k)$$

$$J(\vec{x}_k)(\vec{x} - \vec{x}_k) = -\vec{f}(\vec{x}_k)$$

$$J(\vec{x}_k)\vec{s}_k = -\vec{f}(\vec{x}_k)$$

$\vec{s}_k$ ← unknown

$$\vec{x} - \vec{x}_k = -\left[J(\vec{x}_k)\right]^{-1}\vec{f}(\vec{x}_k)$$

$\vec{x} - \vec{x}_k = \vec{s}_k$

$$\vec{x}_{k+1} = \vec{x}_k - \left[J(\vec{x}_k)\right]^{-1}\vec{f}(\vec{x}_k)$$

Matrix inversion

In practice, use "\"

```
s = -J(x) \ f(x);
```

cf) $x_{k+1} = x_k - \dfrac{f(x_k)}{f'(x_k)}$

# The Multidimensional Newton's Method (cont')

- In practice, we do not compute $\mathbf{J}^{-1}$. Rather, the $k$th Newton step $\mathbf{s}_k = x_{k+1} - x_k$ is found by solving the square linear system

$$\mathbf{J}(\mathbf{x}_k)\mathbf{s}_k = -\mathbf{f}(\mathbf{x}_k),$$

  which is solved using the backslash in MATLAB.

- Suppose `f` and `J` are MATLAB functions calculating $\mathbf{f}$ and $\mathbf{J}$, respectively. Then the Newton iteration is done simply by

```
% x is a Newton iterate (a column vector).
% The following is the key fragment
% inside Newton iteration loop.
fx = f(x)
s = -J(x)\fx;
x = x + s;
```

residual ← `fx = f(x)`

Newton step ← `s = -J(x)\fx;`

`x = x + s;` → $x = x - J(x) \backslash f(x)$ ;

$\vec{x}_{k+1} = \vec{x}_k - [J(\vec{x}_k)]^{-1} \vec{f}(\vec{x}_k)$

- Since $\mathbf{f}(x_k)$ is the residual and $\mathbf{s}_k$ is the gap between two consecutive iterates at the $k$th step, monitor their norms to determine when to stop iteration.

# Computer Illustration

~~Let's find a root of the function introduced in the example on p. 5.~~

1. Define $\mathbf{f}$ and $\mathbf{J}$, either as anonymous functions or as function m-files.

```
f = @(x) [exp(x(2)-x(1)) - 2;
          x(1)*x(2) + x(3);
          x(2)*x(3) + x(1)^2 - x(2)];
J = @(x) [-exp(x(2)-x(1)),exp(x(2)-x(1)), 0;
          x(2), x(1), 1;
          2*x(1), x(3)-1, x(2)];
```

1. Define an initial iterate $\mathrm{x}$, say $\mathbf{x}_0 = (0, 0, 0)^{\mathrm{T}}$.

```
x = [0 0 0]';
```

1. Iterate.

```
for k = 1:7
    s = -J(x)\f(x);
    x = x + s;
end
```

# Implementation

(from FNC)

needed for #5 of HW07.

```
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

Newton step

Stopping criteria

(using \)

## Notes

1. This program expects "f" to produce both $\vec{f}$ and J

2. Inevitably, "f" must be defined using a func.m-file

```
function [f,J] = foo(x)
    ⋮
end
```

3. When passing "f" (written in "foo.m") into the program,

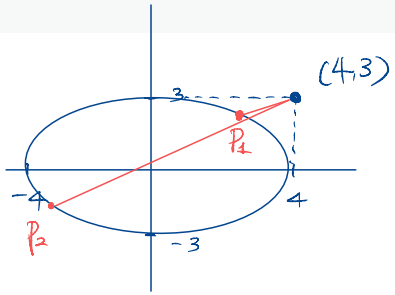
```
x = newtonsys(@foo, x1)
```

Example (cf #5 of HW07)

Find the points on the ellipse

$$\frac{x^2}{16} + \frac{y^2}{9} = 1$$

that are closest and farthest from $(4,3)$.




- Objective function: $f(x,y) = (x-4)^2 + (y-3)^2$ (squared distance)
- Constraint: $g(x,y) = \frac{x^2}{16} + \frac{y^2}{9} = 1$

## Lagrange multiplier

Unknown

$x, y, \lambda$

- $\nabla f = \lambda \nabla g$ (2 eqns)
- $g(x,y) = 1$ (1 eqn)

$\vec{f}(\vec{u}) = \vec{0}$, $\vec{u} = \begin{bmatrix} x \\ y \\ \lambda \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$

$$\begin{cases} 2(x-4) = 2\lambda\, x/16 \\ 2(y-3) = 2\lambda\, y/9 \\ x^2/16 + y^2/9 = 1 \end{cases} \Rightarrow \begin{cases} (1-\lambda/16)x - 4 = 0 \\ (1-\lambda/9)y - 3 = 0 \\ x^2/16 + y^2/9 - 1 = 0 \end{cases}$$

$$\Rightarrow \begin{cases} (1 - u_3/16)\,u_1 - 4 = 0 \\ (1 - u_3/9)\,u_2 - 3 = 0 \\ u_1^2/16 + u_2^2/9 - 1 = 0 \end{cases} : \vec{f}(\vec{u}) = \begin{bmatrix} f_1(u_1,u_2,u_3) \\ f_2(u_1,u_2,u_3) \\ f_3(u_1,u_2,u_3) \end{bmatrix}$$