# Multidimensional Rootfinding

## Rootfinding (scalar)

- Given: $f: \mathbb{R} \to \mathbb{R}$
- Want: $r \in \mathbb{R}$ s.t. $f(r) = 0$

a root or zero of $f$

## Iteration methods

① Fixed point iteration

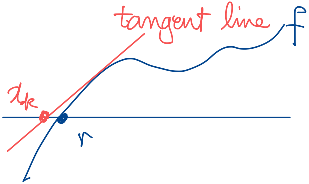
$$g(r) = r \quad (g(x) = x - f(x))$$

- $\begin{cases} x_0 & \text{initial guess} \\ x_{k+1} = g(x_k), & k = 0, 1, 2, \cdots \end{cases}$
- converges if $|g'(r)| < 1$
- linear convergence $(\epsilon_{k+1} \approx g'(r)\, \epsilon_k)$

linear

② Newton's method

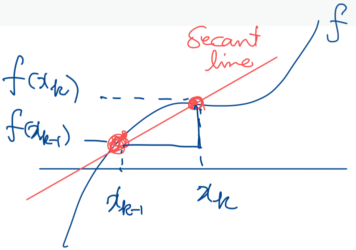
- $\boxed{\text{Idea}}$: Replace $f$ by a simpler function and find its root.




- $\begin{cases} x_0 & \text{initial guess} \\ x_{k+1} = \underbrace{x_k - \dfrac{f(x_k)}{f'(x_k)}}_{\text{form. for } x\text{-intercept of t-line.}}, & k = 0, 1, 2, \cdots \end{cases}$

- quad. convergence:

$$\epsilon_{k+1} \approx \frac{f''(r)}{2f'(r)} \epsilon_k^2$$




linear

## ③ Secant method

- $\boxed{\text{Idea}}$: Replace $f$ by a simpler function and look for its root.

- $\begin{cases} x_{-1}, \; x_0 \quad \text{initial guesses} \\ x_{k+1} = x_k - \dfrac{f(x_k)}{(\text{slope})} = x_k - \dfrac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} f(x_k), \quad k = 0, 1, 2, \dots \end{cases}$

- Superlinear Conv. (but slower than quad.)

$$\epsilon_{k+1} \approx \left( \frac{f''(r)}{2f'(r)} \right)^{\alpha - 1} \epsilon_k^{\alpha} \quad \text{where} \quad \alpha = \frac{1+\sqrt{5}}{2} = 1.6\dots$$

(the golden ratio)

Example (Computation of $\sqrt{a}$ using Newton) Assume $a > 0$.

- Note that $\sqrt{a}$ is a root of $f(x) = x^2 - a$.

$$\downarrow$$

$$f'(x) = 2x$$

$$f(x) = 0$$
$$x^2 - a = 0$$
$$x^2 = a$$
$$x = \pm\sqrt{a}$$

- Newton's iter. formula:

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

$$= x_k - \frac{x_k^2 - a}{2x_k}$$

$$= x_k - \frac{1}{2}x_k + \frac{a}{2x_k} = \frac{1}{2}x_k + \frac{a}{2x_k}.$$

# Newton's Method for Nonlinear Systems

# Multidimensional Rootfinding Problem

## Rootfinding Problem: Vector Version

Given a continuous vector-valued function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$, find a vector $\mathbf{r} \in \mathbb{R}^n$ such that $\mathbf{f}(\mathbf{r}) = \mathbf{0}$.

The rootfinding problem $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ is equivalent to solving the *nonlinear* system of $n$ scalar equations in $n$ unknowns:

$$\vec{f} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}$$

$$\begin{cases} f_1(x_1, \ldots, x_n) = 0, \\ f_2(x_1, \ldots, x_n) = 0, \\ \vdots \\ f_n(x_1, \ldots, x_n) = 0. \end{cases}$$

$f_1, \ldots, f_n$ are scalar functions.

# Multidimensional Taylor Series

cf) $f(x+h) = f(x) + \boxed{f'(x)} h + O(h^2)$

If $\mathbf{f}$ is differentiable, we can write

"Multidimensional analog of derivative"

$$\mathbf{f}(\mathbf{x} + \mathbf{h}) = \mathbf{f}(\mathbf{x}) + \boxed{\mathbf{J}(\mathbf{x})}\mathbf{h} + O(\|\mathbf{h}\|^2),$$

center perturbation

where $\mathbf{J}$ is the **Jacobian matrix** of $\mathbf{f}$

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} = \left[\frac{\partial f_i}{\partial x_j}\right]_{i,j=1,\ldots,n}.$$

$\vec{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad \vec{h} = \begin{bmatrix} h_1 \\ \vdots \\ h_n \end{bmatrix}$

- The first two terms $\mathbf{f}(\mathbf{x}) + \mathbf{J}(\mathbf{x})\mathbf{h}$ is the "linear approximation" of $\mathbf{f}$ near $\mathbf{x}$.
- If $\mathbf{f}$ is actually linear, *i.e.*, $\mathbf{f}(\mathbf{x}) = A\mathbf{x} - \mathbf{b}$, then the Jacobian matrix is the coefficient matrix $A$ and the rootfinding problem $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ is simply $A\mathbf{x} = \mathbf{b}$.

# Example

Let

$$
\begin{aligned}
f_1(x_1, x_2, x_3) &= -x_1 \cos(x_2) - 1, \\
f_2(x_1, x_2, x_3) &= x_1 x_2 + x_3, \\
f_3(x_1, x_2, x_3) &= e^{-x_3} \sin(x_1 + x_2) + x_1^2 - x_2^2.
\end{aligned}
$$

Then

$$
\mathbf{J}(\mathbf{x}) = \begin{bmatrix} -\cos(x_2) & x_1 \sin(x_2) & 0 \\ x_2 & x_1 & 1 \\ e^{-x_3} \cos(x_1 + x_2) + 2x_1 & e^{-x_3} \cos(x_1 + x_2) - 2x_2 & -e^{-x_3} \sin(x_1 + x_2) \end{bmatrix}.
$$

**Exercise.** Write out the linear part of the Taylor expansion of

$$
f_1(x_1 + h_1, x_2 + h_2, x_3 + h_3), \quad \text{near } (x_1, x_2, x_3).
$$

# The Multidimensional Newton's Method

Recall the idea of Newton's method:

> *If finding a zero of a function is difficult, replace the function with a simpler approximation (linear) whose zeros are easier to find.*

Applying the principle:

- Linearize $\mathbf{f}$ at the $k$th iterate $\mathbf{x}_k$:

$$\mathbf{f}(\mathbf{x}) \approx L(\mathbf{x}) = \mathbf{f}(\mathbf{x}_k) + \mathbf{J}(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k).$$

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

- Define the next iterate $\mathbf{x}_{k+1}$ by solving $L(\mathbf{x}_{k+1}) = \mathbf{0}$:

$$\mathbf{0} = \mathbf{f}(\mathbf{x}_k) + \mathbf{J}(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) \quad \Longrightarrow \quad \mathbf{x}_{k+1} = \mathbf{x}_k - \left[\mathbf{J}(\mathbf{x}_k)\right]^{-1} \mathbf{f}(\mathbf{x}_k).$$

Note that $\mathbf{J}^{-1}\mathbf{f}$ plays the same role as $f/f'$ in the scalar Newton.

# 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;
```

- 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

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)];
```

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

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

3. Iterate.

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

# Implementation

```
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
```