Computational Physics

Shuai Dong

Isaac Newton

Gottfried Leibniz

Numerical calculus

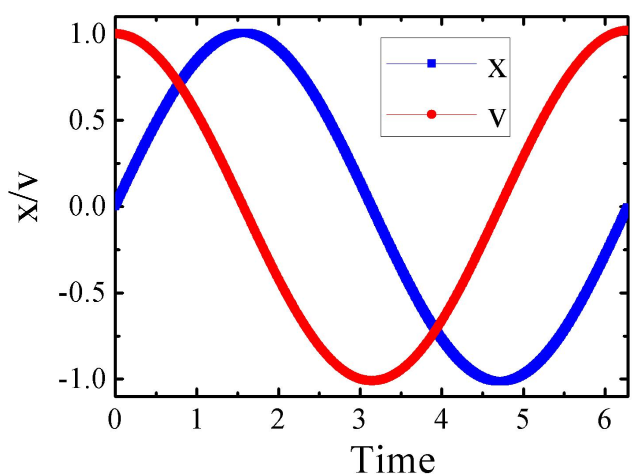
x
position

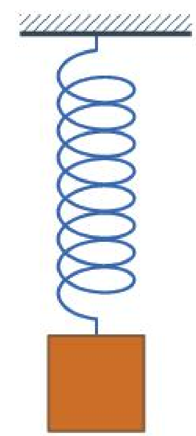
derivative →
← integral

v
velocity

derivative →
← integral

a
acceleration





Numerical calculus

- Numerical differentiation
- Numerical integration
- Roots of an equation
- Extremes of a function

Numerical differentiation

Taylor expansion:

single-variable:

$$f(x) = f(x_0) + (x - x_0) f'(x_0) + \frac{(x - x_0)^2}{2!} f''(x_0) + \cdots + \frac{(x - x_0)^n}{n!} f^{(n)}(x_0) + \cdots$$

multivariable:

$$f(x, y, \cdots) = f(x_0, y_0, \cdots) + (x - x_0) f_x(x_0, y_0, \cdots) + (y - y_0) f_y(x_0, y_0, \cdots) + \cdots \frac{(x - x_0)^2}{2!} f_{xx}(x_0, y_0, \cdots) + \frac{(y - y_0)^2}{2!} f_{yy}(x_0, y_0, \cdots) + \frac{2(x - x_0)(y - y_0)}{2!} f_{xy}(x_0, y_0, \cdots) + \cdots$$

To calculate f', f'', f''' .... $f_{xy} = \partial^2 f / \partial x \partial y$

The first-order derivative of a single-variable function f (x) around a point x_i is defined from the limit.

$$f'(x_i) = \lim_{\Delta x \to 0} \frac{f(x_i + \Delta x) - f(x_i)}{\Delta x}$$

Now if we divide the space into discrete points x_i with evenly spaced intervals $x_{i+1} - x_i = h$ and label the function at the lattice points as $f_i = f(x_i)$, we obtain the simplest expression for the first-order derivative.

the two-point formula for the first-order derivative	$f_i' = \dfrac{f_{i+1} - f_i}{h} + O(h)$

An improved choice:

$$f_{i+1} = f_i + h \cdot f_i' + h^2 \cdot f_i''/2 + h^3 \cdot f_i'''/6 + \cdots$$

$$f_{i-1} = f_i - h \cdot f_i' + h^2 \cdot f_i''/2 - h^3 \cdot f_i'''/6 + \cdots$$

$$\Rightarrow f_{i+1} - f_{i-1} = 0 + 2h \cdot f_i' + 0 + h^3 \cdot f_i'''/3 + \cdots$$

$$f_{i+1} - f_{i-1} = 2h \cdot f_i' + \mathrm{O}(h^3)$$

The accuracy is improved from O(h) to O(h²).

The three-point formula for the first-order derivative

$$f_i' = \frac{f_{i+1} - f_{i-1}}{2h} + \mathrm{O}(h^2)$$

A five-point formula can be derived by including the expansions of f_{i+2} and f_{i-2} around x_i.

$$f_{i+1} - f_{i-1} = 2hf_i' + \frac{h^3}{3} f_i^{(3)} + O(h^5) \qquad (1)$$

$$f_{i+2} - f_{i-2} = 4hf_i' + \frac{8h^3}{3} f_i^{(3)} + O(h^5) \qquad (2)$$

$$(1) \times 8 - (2)$$

$$\Rightarrow f_i' = \frac{1}{12h}(f_{i-2} - 8f_{i-1} + 8f_{i+1} - f_{i+2}) + O(h^4)$$

Summary

number of points	inaccuracy
2	O(h)
3	O(h²)
5	O(h⁴)

More points

Smaller h

Higher accuracy

The three-point formula for the second-order derivative

$$f_{i+1} = f_i + h \cdot f_i' + h^2 \cdot f_i''/2 + h^3 \cdot f_i'''/6 + \cdots$$

$$f_{i-1} = f_i - h \cdot f_i' + h^2 \cdot f_i''/2 - h^3 \cdot f_i'''/6 + \cdots$$

$$f_{i+1} - 2f_i + f_{i-1} = h^2 f_i'' + O(h^4)$$

$$f_i'' = \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2} + O(h^2)$$

Similarly, we can combine the expansions of $f_{i\pm2}$ and $f_{i\pm1}$ around x_i and f_i to cancel the f'_i, $f^{(3)}_i$, $f^{(4)}_i$, and $f^{(5)}_i$ terms; then we have

$$f_i'' = \frac{1}{12h^2}(-f_{i-2} + 16f_{i-1} - 30f_i + 16f_{i+1} - f_{i+2}) + O(h^4)$$

the five-point formula for the second-order derivative

Example

- Given $f(x)=\sin(x)$, let's calculate $f'(x)$ & $f''(x)$.
- Divide the region from 0 to $\pi/2$ to 100 equal-length intervals with 101 points $i*\pi/200$ ($i=0,1,2......100$).
- For boundary points ($i=0,1$ & $99,100$), we can use Lagrange interpolation to extrapolate the derivatives.

Code example:

3.1.Differentiation.cpp

Three-point formula for f'
Three-point formula for f''

Numerical calculus

- Numerical differentiation
- **Numerical integration**
- Roots of an equation
- Extremes of a function

Numerical integration

In general, we want to obtain the numerical value of an integral, defined in the region [a, b],

$$S = \int_a^b f(x)dx.$$

Divide the region [a, b] into n slices, evenly spaced with an interval h. If we label the data points as x_i with i = 0, 1, . . . , n, we can write the entire integral as a summation of integrals, with each over an individual slice.

$$\int_a^b f(x)\,dx = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)\,dx$$

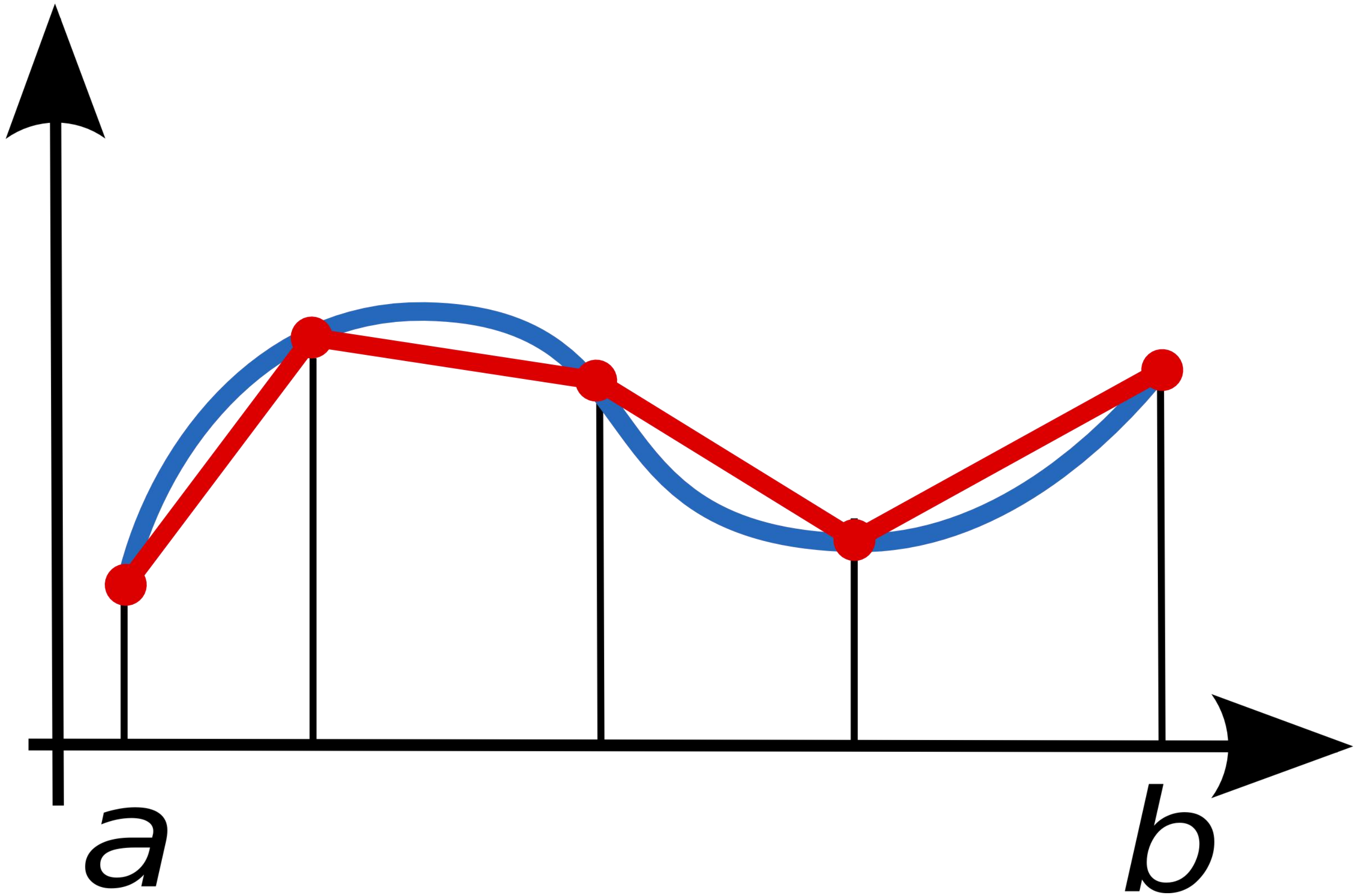
The simplest quadrature is obtained if we approximate f (x) in the region [x_i , x_{i+1}] linearly, that is,

$$f(x) \approx f_i + (x - x_i)(f_{i+1} - f_i)/h$$

$$S = \frac{h}{2}\sum_{i=0}^{n-1}(f_i + f_{i+1}) + O(h^2)$$

The above quadrature is commonly referred as the trapezoidal rule, which has an overall accuracy up to $O(h^2)$.

Trapezoidal rule



We can obtain a quadrature with a higher accuracy by working on two slices together. If we apply the Lagrange interpolation to the function f (x) in the region [x_{i-1}, x_{i+1}], we have

$$f(x) = \frac{(x - x_i)(x - x_{i+1})}{(x_{i-1} - x_i)(x_{i-1} - x_{i+1})} f_{i-1} + \frac{(x - x_{i-1})(x - x_{i+1})}{(x_i - x_{i-1})(x_i - x_{i+1})} f_i$$

$$+ \frac{(x - x_{i-1})(x - x_i)}{(x_{i+1} - x_{i-1})(x_{i+1} - x_i)} f_{i+1} + O(h^3)$$

$$S = \frac{h}{3} \sum_{j=0}^{n/2-1} (f_{2j} + 4 f_{2j+1} + f_{2j+2}) + O(h^4)$$

Example

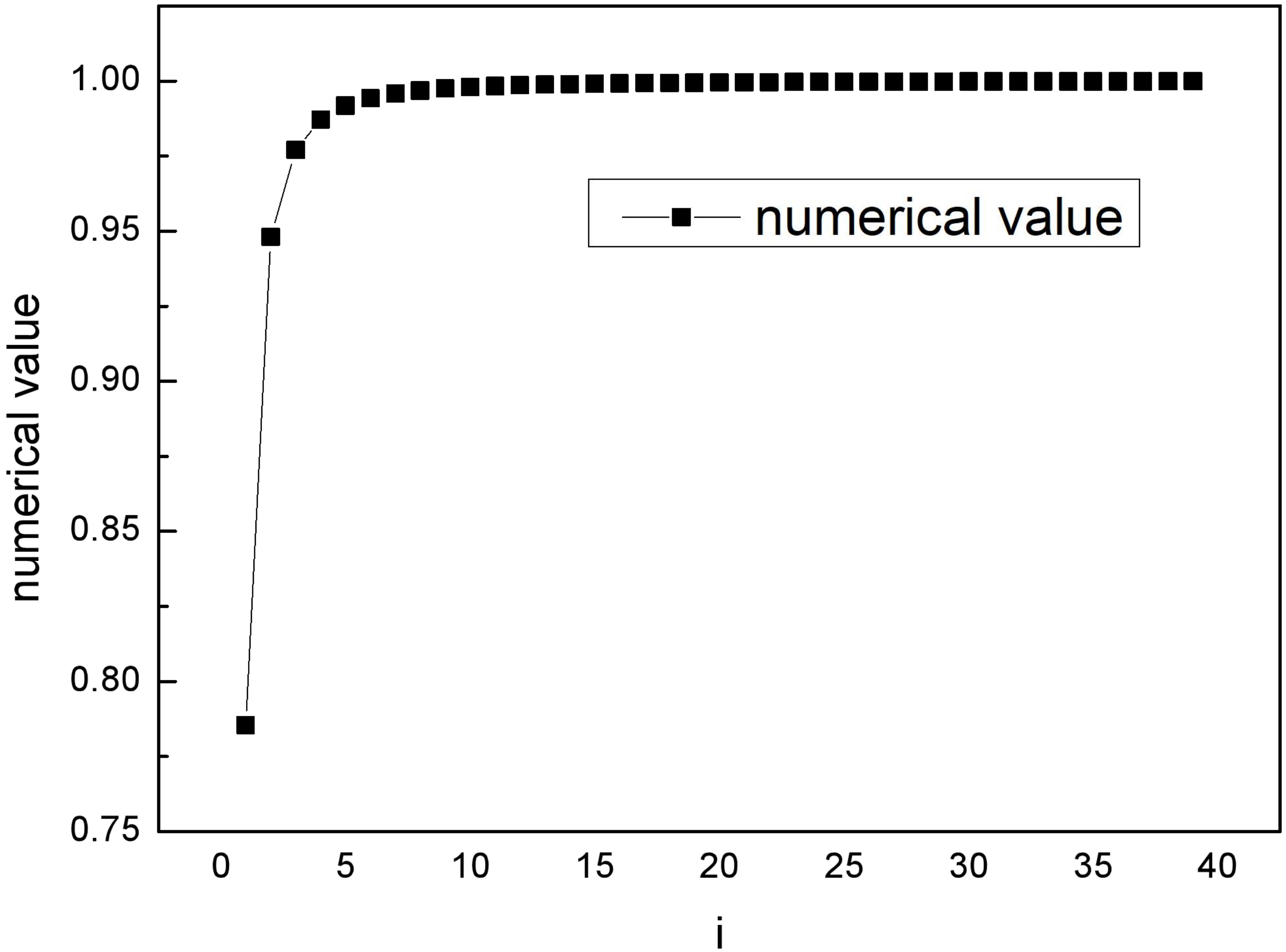
- Given $f=\sin(x)$, integrate f from 0 to $\pi/2$.

- The analytic function: $-\cos(x)$
- The exact value: $\cos(0)-\cos(\pi/2)=1$.

- We can use trapezoidal rule to see how the numerical value converges to 1.

Code example

- [3.2.Integration.cpp]



Homework

- Improved integration with the three-point Lagrange interpolation implemented.
- Comparison with the trapezoidal rule method.

$$S = \frac{h}{3}\sum_{j=0}^{n/2-1}(f_{2j} + 4f_{2j+1} + f_{2j+2}) + O(h^4)$$

Numerical calculus

- Numerical differentiation
- Numerical integration
- Roots of an equation
- Extremes of a function

Roots of an equation

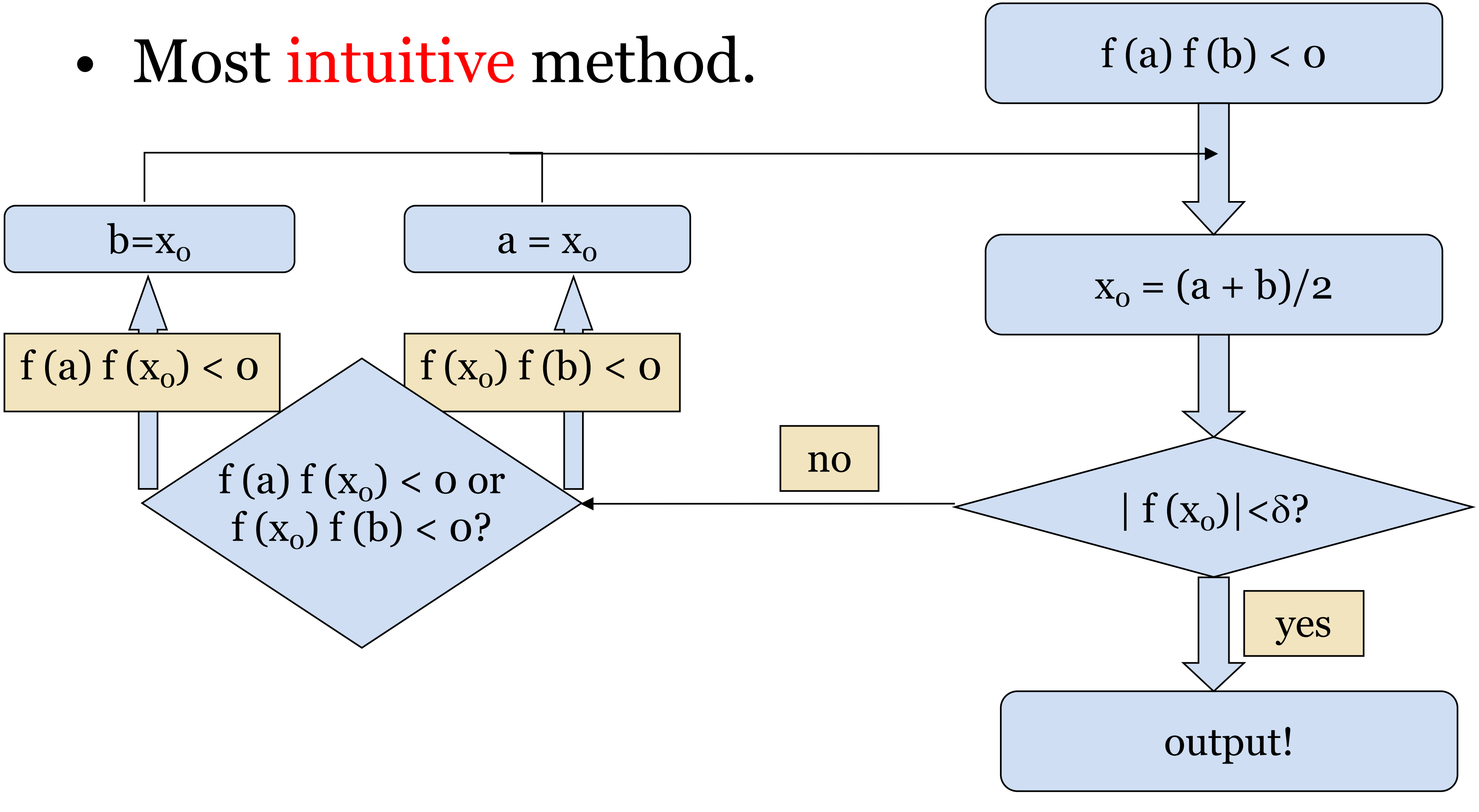
In physics, we often encounter situations in which we need to find the possible value of x that ensures the equation $f(x)=0$, where $f(x)$ can either be an explicit or an implicit function of x. If such a value exists, we call it a root or zero of the equation.

If we need to find a root for $f(x)=a$, then how?

define $g(x)=f(x)-a$, and find a root for $g(x)=0$.

Bisection method

- If we know that there is a root x_r in the region [a,b] for f(x)=0, we can use the bisection method to find it within a required accuracy.
- Most intuitive method.



Code Example

- $f(x)=\sin(x)=0.5$; x is within 0 to $\pi/2$.
- Analytically, we know the root is $\pi/6$.
- Numerically, the procedure is:

  since $[\sin(0)-0.5]*[\sin(\pi/2)-0.5]<0$ and $[\sin(0)-0.5]*[\sin(\pi/4)-0.5]<0$, but $[\sin(\pi/2)-0.5]*[\sin(\pi/4)-0.5]>0$;
- then the root must be within $(0,\pi/4)$.
- Then we calculate the value at $\pi/8$.
- ……
- 3.3.Bisection.cpp

The Newton method

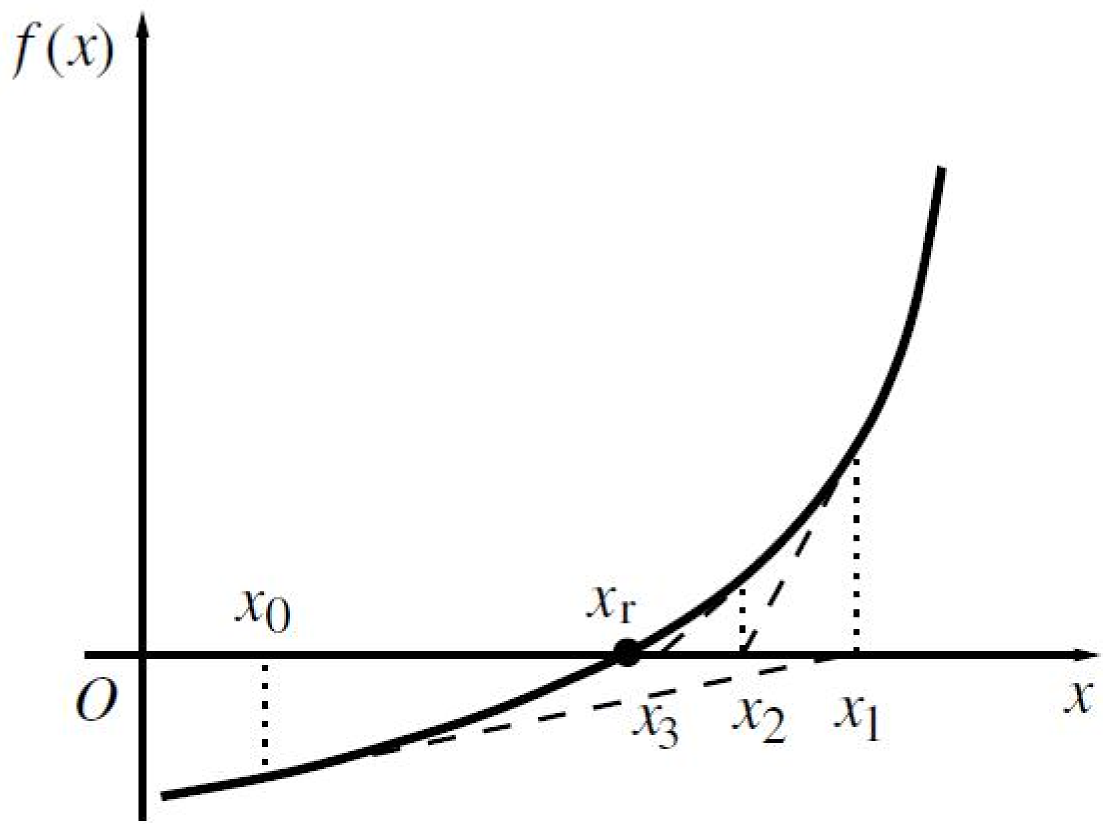
This method is based on linear approximation of a smooth function around its root. We can formally expand the function $f(x_r) = 0$ in the neighborhood of the root x_r through the Taylor expansion.

$$f(x_r) \approx f(x) + (x_r - x) f'(x) + \cdots = 0$$

where x can be viewed as a trial value for the root of x_i at the *i*th step and the approximate value of the next step x_{i+1} can be derived.

$$x_{i+1} = x_i + \Delta x_i = x_i - f_i / f_i'$$

(i = 0, 1, ...)



Code example

Example:

$f(x)=\sin(x)=0.5$; $g(x)=f(x)-0.5=\sin(x)-0.5$

i	x_i	g_i	g_i'
0	0	-0.5	1
1	0.5	……	……

- 3.4.NewtonRoot.cpp

Possible bugs

- If the function is not monotonous
- If $f'_i=0$ or very small at some points
- Works well when the function is monotonous, especially with moderate f'.

Secant method
- discrete Newton method

In many cases, especially when f (x) has an implicit dependence on x, an analytic expression for the first-order derivative needed in the Newton method may not exist or may be very difficult to obtain.

We have to find an alternative scheme to achieve a similar algorithm. One way to do this is to replace the analytic f'(x) with the two-point formula for the first-order derivative, which gives

$$x_{i+1} = x_i - (x_i - x_{i-1}) f_i / (f_i - f_{i-1})$$

Code example

Example:

f(x)=sin(x)=0.5; g(x)=f(x)-0.5=sin(x)-0.5

i	x_i	g_i
0	0	-0.5
1	π/2	0.5
2	π/4	……

$$x_{i+1} = x_i - (x_i - x_{i-1})g_i / (g_i - g_{i-1})$$

$$x_2 = \frac{\pi}{2} - (\frac{\pi}{2} - 0) \cdot 0.5 / (0.5 + 0.5) = \frac{\pi}{4}$$

3.5.Secant .cpp

Numerical calculus

- Numerical differentiation
- Numerical integration
- Roots of an equation
- **Extremes of a function**

Extremes of a function

- An associated problem to find the root of an equation is finding the maxima and/or minima of a function.
- Examples of such situations in physics occur when considering the equilibrium position of an object, the potential surface of a field, and the optimized structures of molecules and small clusters.

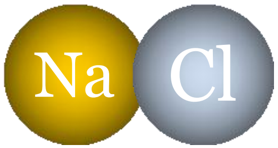
- We know that an extreme of g(x) occurs at the point with

$$f(x) = \frac{dg(x)}{dx} = 0$$

which is a minimum (maximum) if $f'(x) = g''(x)$ is greater (less) than zero. So all the root-search schemes discussed so far can be generalized here to search for the extremes of a single-variable function.

Example

The (ionic) bond length of the diatomic molecule



$$V(r) = -\frac{e^2}{4\pi\varepsilon_0 r} + V_0 \exp(-\frac{r}{r_0})$$

where e is the charge of a proton, ε_o is the electric permittivity of vacuum, and V_o and r_o are parameters of this effective interaction.

The first term comes from the Coulomb interaction between the two ions, but the second term is the result of the electron distribution in the system.

The force:

$$f(r) = -\frac{dV(r)}{dr} = -\frac{e^2}{4\pi\varepsilon_0 r^2} + \frac{V_0}{r_0}\exp(-\frac{r}{r_0})$$

At equilibrium, the force between the two ions is zero. Therefore, we search for the root of f (x) = -dV(x)/dx = 0.

code example

- parameters for NaCl
- $e^2/4\pi\varepsilon_o = 14.4$ AeV
- $V_o = 1.09 \times 10^3$ eV
- $r_o = 0.33$ A

- r starts from 1 A

- 3.6.NaCl.cpp

In the example program above, the search process is forced to move along the direction of descending the function g(x) when looking for a minimum. In other words, for $x_{i+1}=x_i+\Delta x_i$, the increment Δx_i has the sign opposite to $g'(x_i)$. Based on this observation, an update scheme can be formulated:

$$\Delta x_i = -f_i / f_i' \Longrightarrow \Delta x_i = -a \cdot g'_i = -a \cdot f_i$$

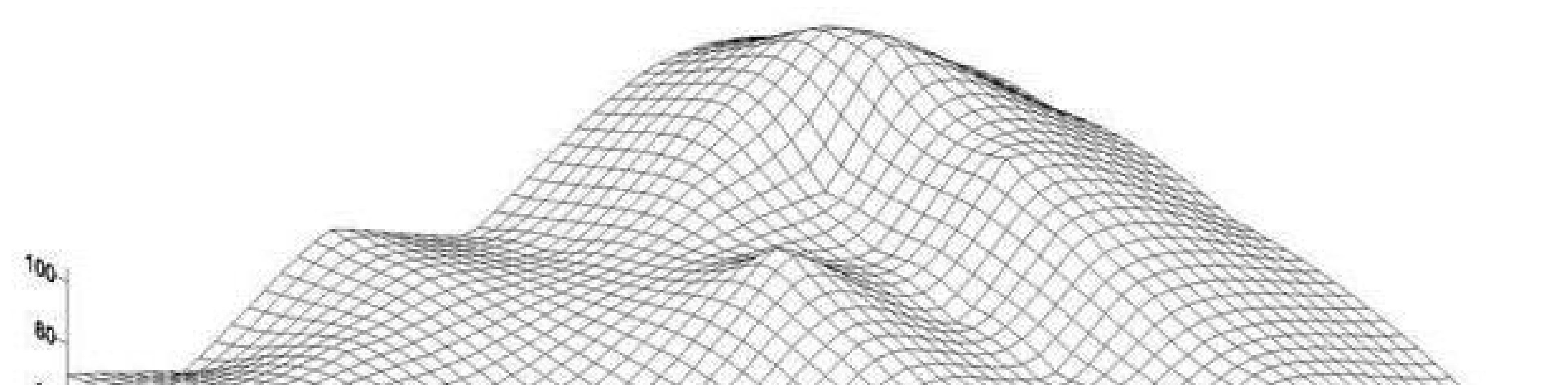
with 'a' being a positive, small, and adjustable parameter. For the minimum, f' (or g'') must be positive.

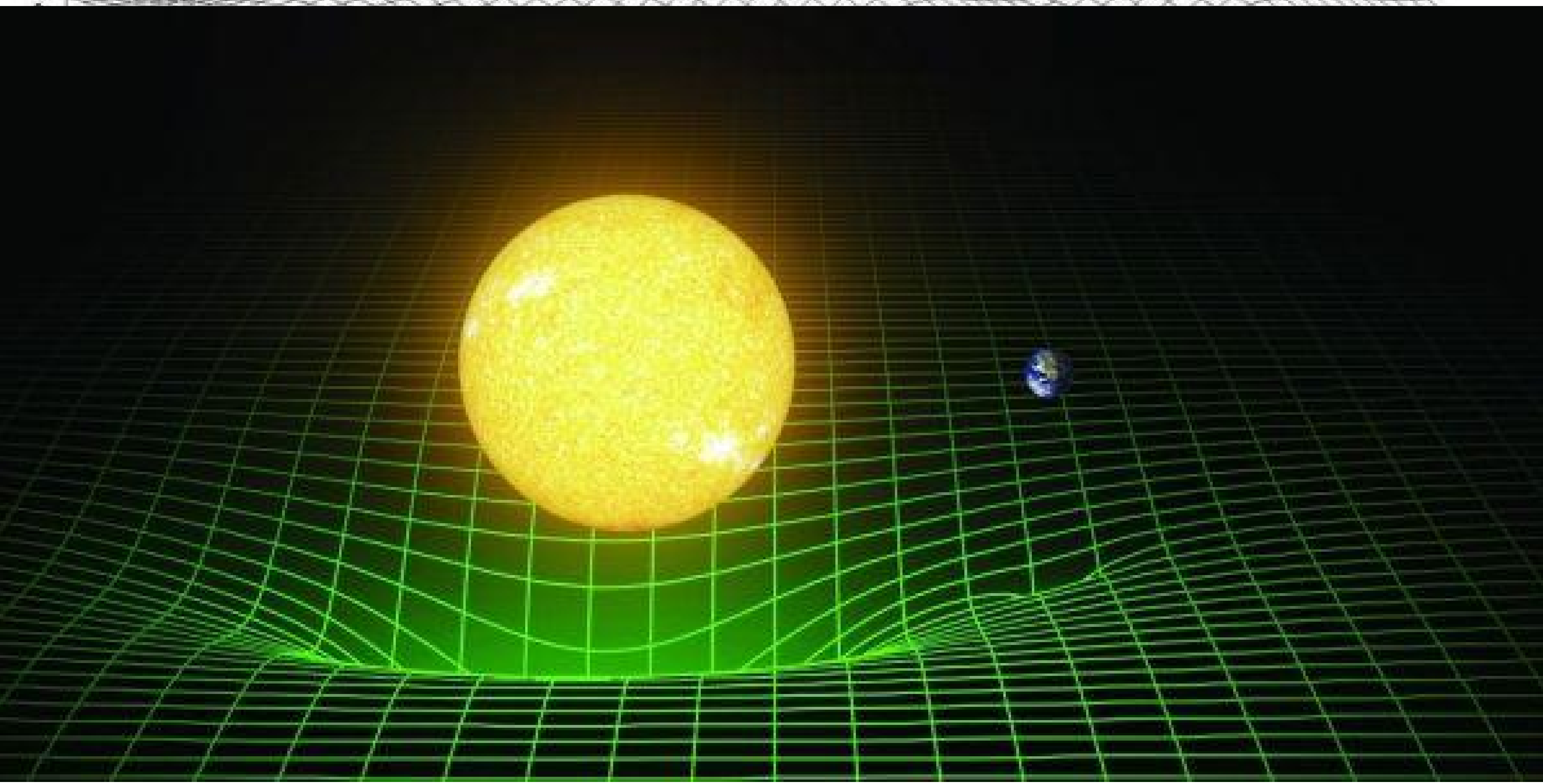
This scheme can be generalized to the multivariable case as

$$x_{i+1} = x_i + \Delta x_i = x_i - a \cdot \nabla g(x_i) / |\nabla g(x_i)|$$

where $x = (x_1, x_2, \ldots, x_l)$ and $\nabla g(x) = (\partial g/\partial x_1, \partial g/\partial x_2, \ldots, \partial g/\partial x_l)$.

Note that step Δx_i here is scaled by $|\nabla g(x_i)|$ and is forced to move toward the direction of the steepest descent. This is why this method is known as the steepest-descent method.


100
80



Homework

Search for the minimum of the function

g(x,y)=sin(x+y)+cos(x+2*y)

in the whole space