Computational Techniques: Root-finding and Optimization of Functions

European Master in Theoretical Chemistry and Computational Modelling

Elena Formoso
University of the Basque Country

December 1, 2021

Universidad del País Vasco Euskal Herriko Unibertsitatea

Schedule

Deadlines

- 10th December: Numerical Integration homework (continuous evaluation, no marks)
- 17th December: Root-finding and function optimization homework (continuous evaluation, no marks)
- 10th March: Evaluation homeworks (Gradabable)

Outline

(Note: we are not going to diagonalize the Hessian)

Outline

Introduction

What is Mathematical Optimization?

Is the mathematical discipline which is concerned with finding and characterizing the singular points of a function $f(x_1, x_2, \cdots, x_n)$, possibly subject to some constraints.

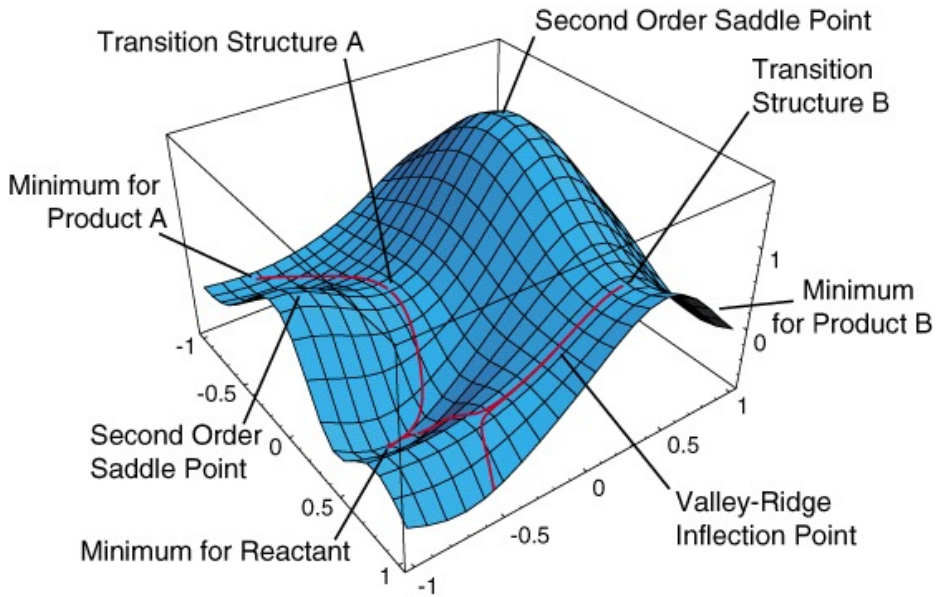
Where can Mathematical Optimization be used?

in Engineering, Economics, Electric Circuits, Transportation, $\cdots$
$\cdots$ and in computational quantum chemistry:

- Variational problems: Electronic structure calculation methods
  HF method (minimization of energy as a function of the orbitals), CASSCF method (minimization of the energy with respect to both the coefficients and the orbitals), $\cdots$.
- Analysis of potential energy surfaces:
  Finding existing minima (reactants, products or intermediates) and saddle points (transition states).

Potential Energy Surface (PES)

Energy as a function of nuclear coordinates.



Usually optimization is applied to very complex problems:

- Multi-dimensional problems.
- Multiple minima (local or global).
- Several order saddle points.

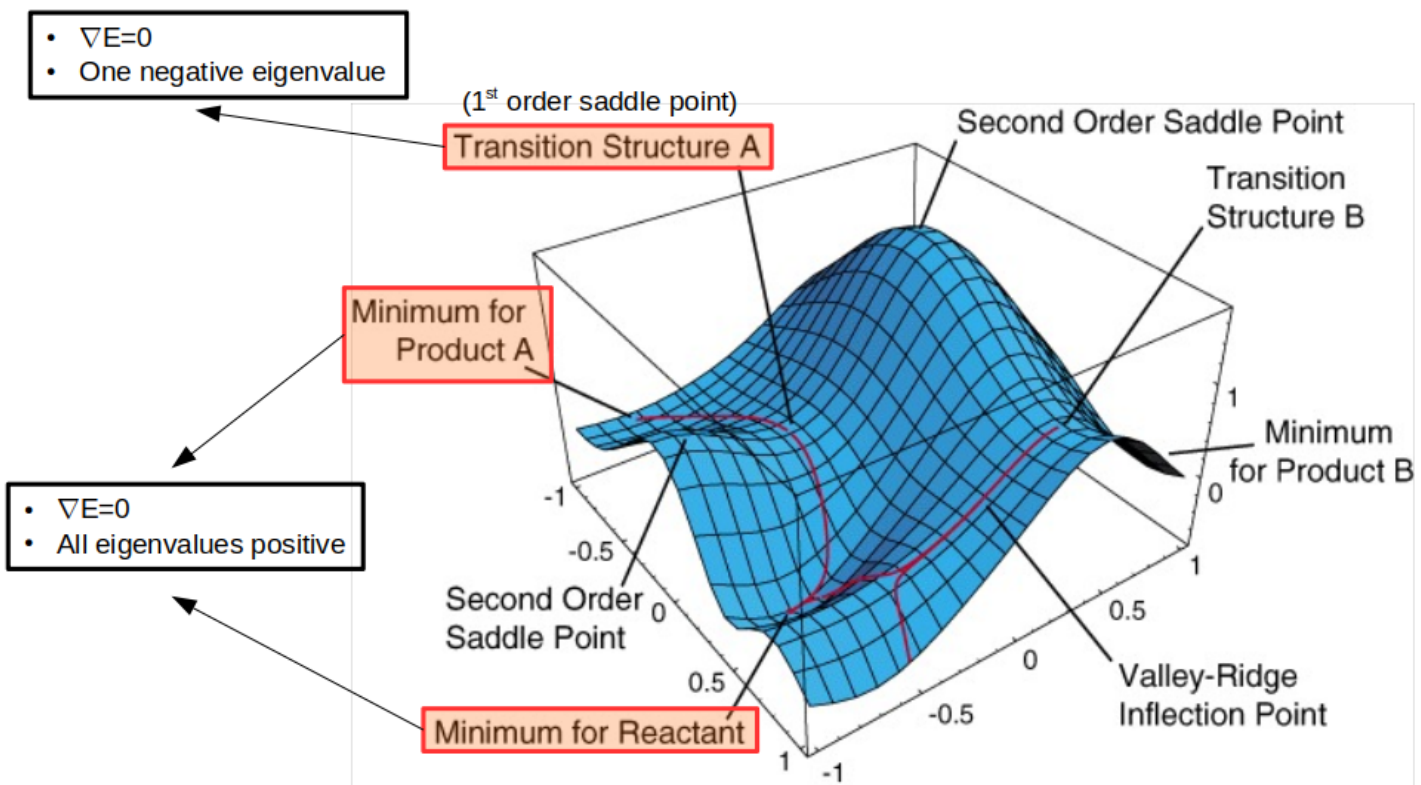
Each evaluation step of the function can be very expensive (sometimes we need to compute first and second derivatives).

How to characterize stationary points

- We need to evaluate:
  - $f(x_1, x_2, \cdots, x_n)$
  - Gradient $= \frac{\partial f}{\partial x_i} = 0; \quad \{i = 1, \cdots n\}$
  - Hessian $\Rightarrow$

$$\begin{pmatrix} \partial f/\partial x_1^2 & \partial f/\partial x_1 \partial x_2 & \partial f/\partial x_1 \partial x_3 & \cdots & \partial f/\partial x_1 \partial x_N \\ & \partial f/\partial x_2^2 & \partial f/\partial x_2 \partial x_3 & \cdots & \partial f/\partial x_2 \partial x_N \\ & & \partial f/\partial x_3^3 & \cdots & \partial f/\partial x_3 \partial x_N \\ & & & \cdots & \cdots \\ & & & & \partial f/\partial x_N^2 \end{pmatrix}$$

- And diagonalize the Hessian to get eigenvalues



Outline

How to find the roots of a function

x_i so that $f(x_i) = 0$

1. Methods based on the Bolzano's Theorem.
2. Newton-Raphson Method.

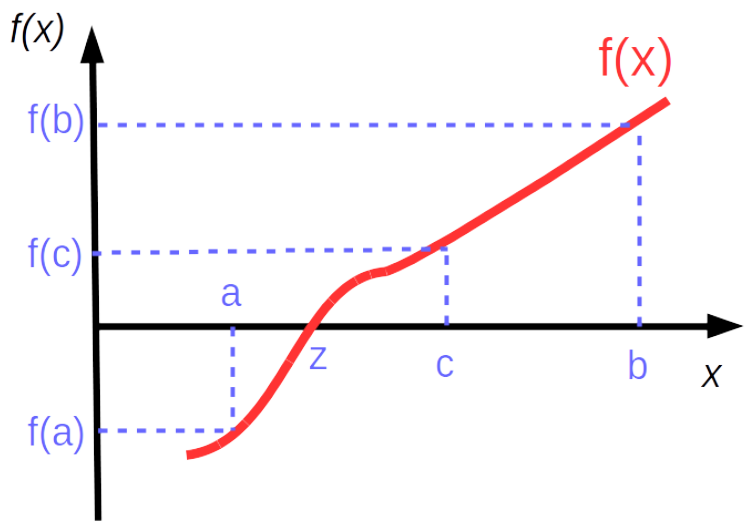
Methods based on **Bolzano's theorem**

Bolzano's theorem

If the function $f(x)$ is continuous and takes positive and negative values in a closed interval [a,b], there is at least a point $z \in [a, b]$ so that $f(z) = 0$.

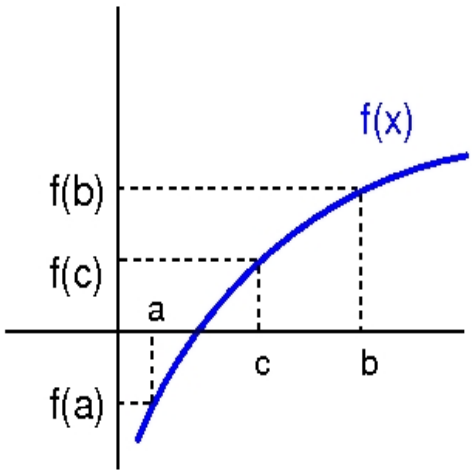
Algorithm:

1. Select an [a,b] interval that fulfils the theorem, $f(a) < 0$ and $f(b) > 0$ (from picture).
2. Choose a value for c, i.e. $c = \frac{a+b}{2}$.
3. Evaluate $f(c)$.
4. IF:
   - $f(c) \approx 0 \Rightarrow$ END program
   - $f(c) > 0$ then root is in interval [a,c], make $b = c$ and GO TO step 2
   - $f(c) < 0$ then root is in interval [c,b], make $a = c$ and GO TO step 2



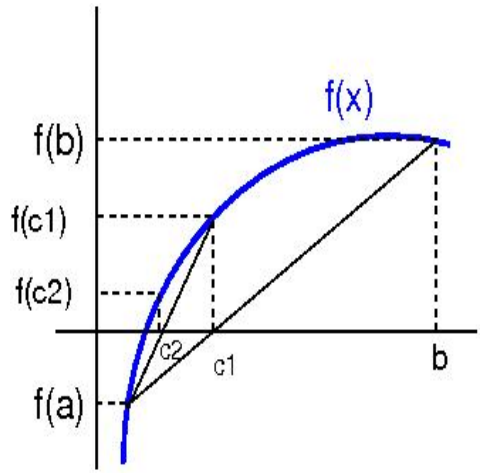
Methods based on **Bolzano's theorem**

Bisection method:



$$c = \frac{a+b}{2}$$

False position method (regula falsi):

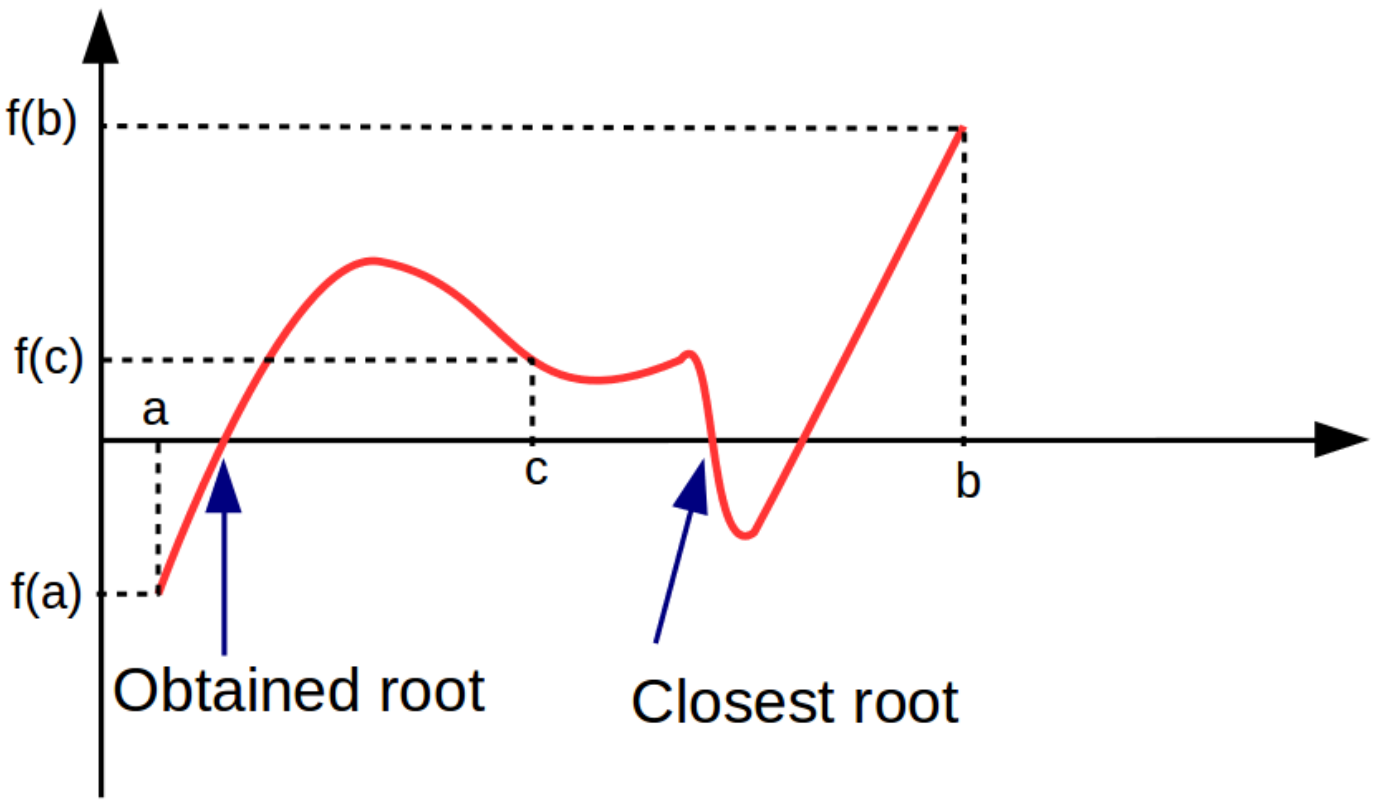


$$c = \frac{af(b) - bf(a)}{f(b) - f(a)}$$

Methods based on **Bolzano's theorem**

Some remarks

- It is not a local method; it does not convergence necessarily to the closest root.
- $df(x)/dx$ (rate of change) not taken into account, only f(x), what may imply lossing better intermediate points.
- Slow convergence, but convergence guaranteed.
- Useful as an initial approximation method.



How to find the roots of a function

x_i so that $f(x_i) = 0$

1. Methods based on the Bolzano's Theorem.
2. Newton-Raphson Method.

Newton-Raphson method (one-dimension)

This is an important procedure in numerical analysis. In its more general form, it can be applied to find the roots of a system of non-linear equations.

- In contrast to the previous methods,
  - it does not requiere a closed [a,b] interval.
  - it requires both the evaluation of the function and the derivative.
- If $f(x)$ is continuous and differentiable, f is approximated at a given point x_0 by its tangent line (first *Taylor polynomial*, ignoring higher order terms):

$$f(x) = f(x_0) + f'(x_0)(x - x_0) \quad \text{Eq. of tangent line}$$

- the x-intercept of this tangent line, $f(x) = 0$, will provide us with x_1 which is a better approximation to the root

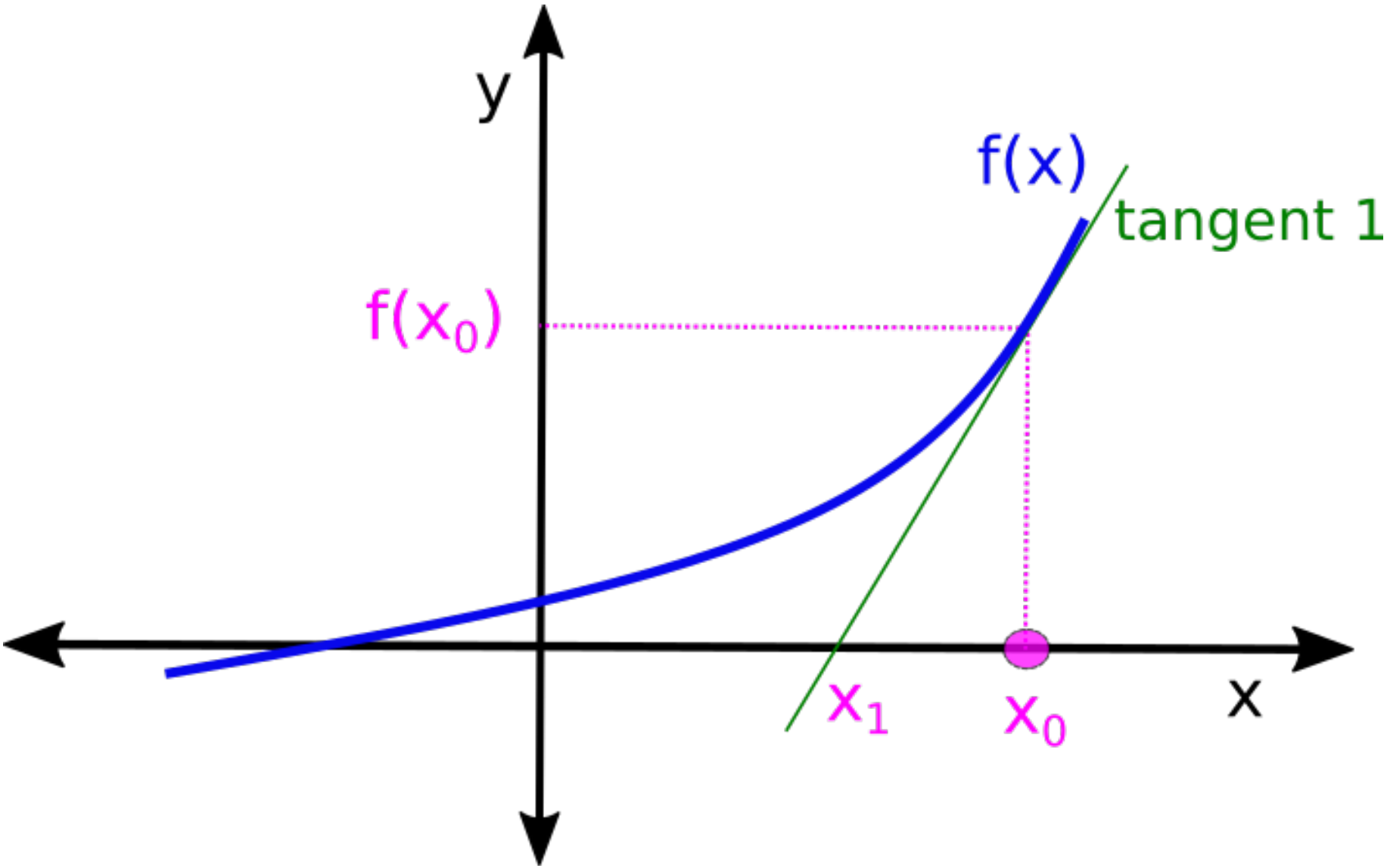
$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

- We repeat the process so that

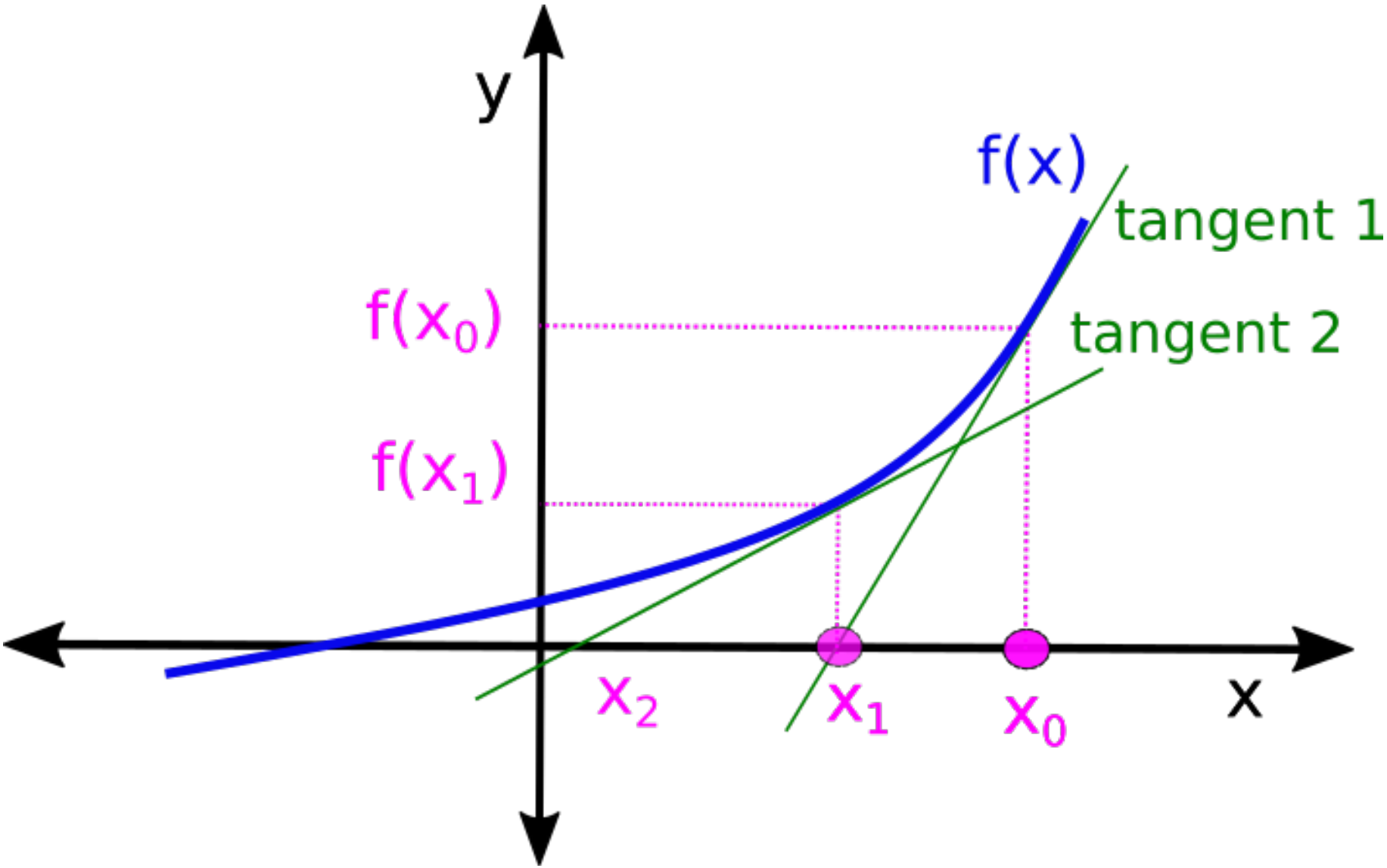
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad \text{iterative procedure}$$

until $x_{n+1} - x_n \leq$ tolerance and $f(x_{n+1}) \leq$ tolerance.

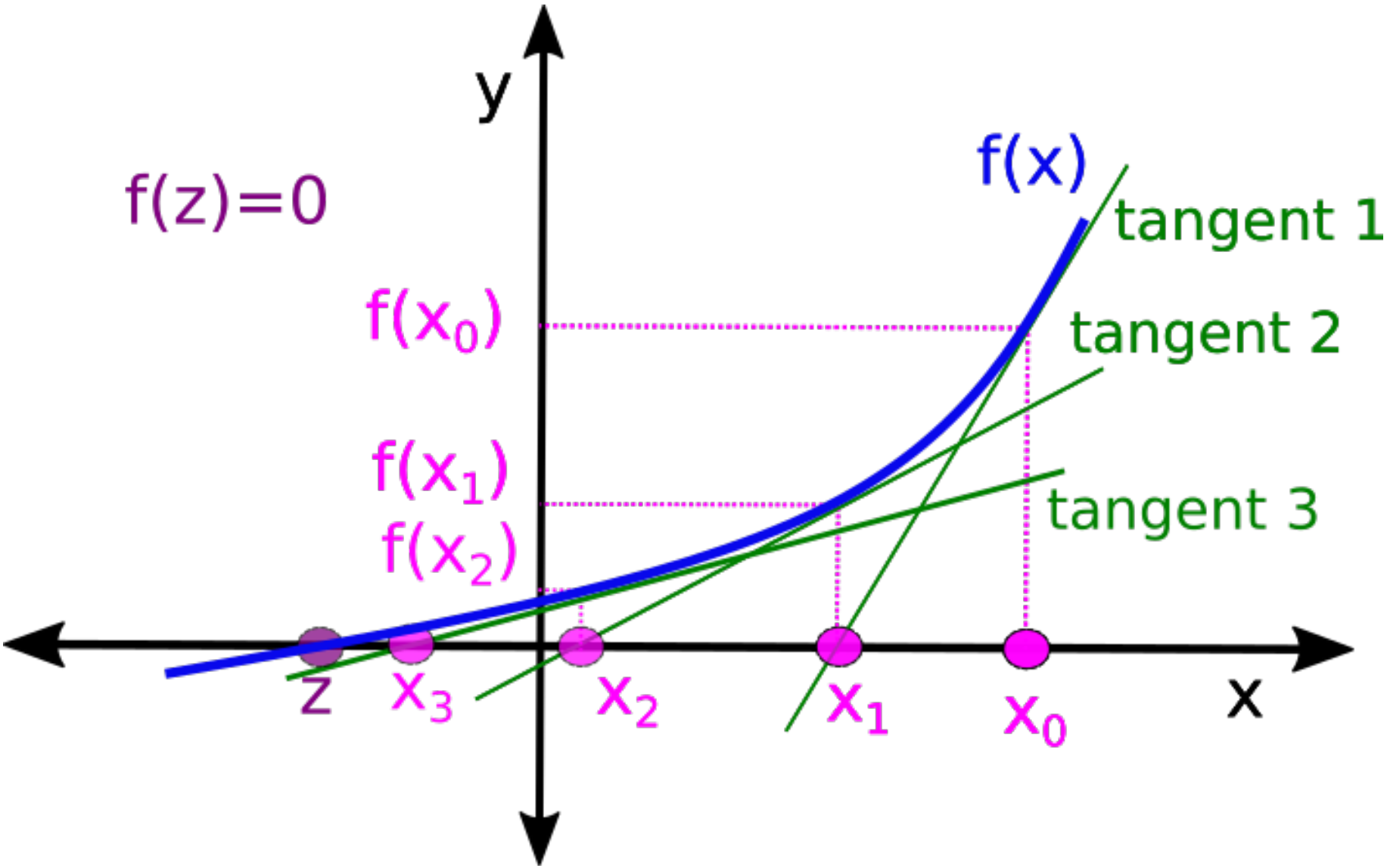
Newton-Raphson iterative method



Newton-Raphson iterative method

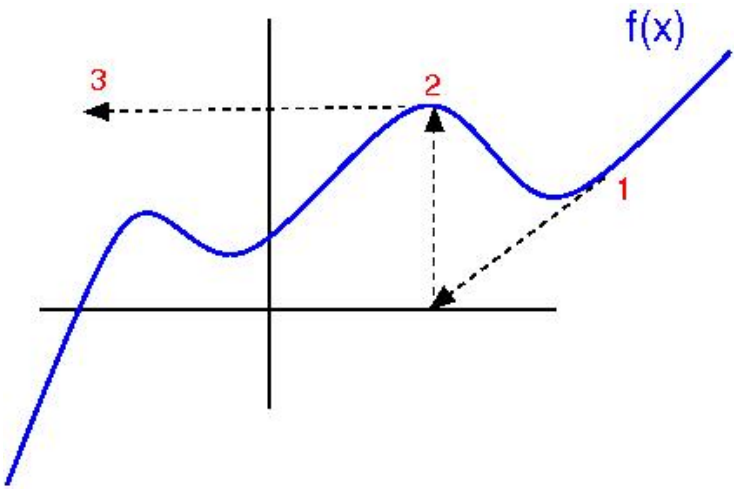


Newton-Raphson iterative method



Newton-Raphson method (one-dimension)

Fail:



$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Important remarks

- If x_0 is sufficiently close to the root/zero (z), then the subsequent points x_n converge to z quadratically.
- Newton-Raphson is a local method: it will find the root closest to the initial point.
- Convergence is not guaranteed. Sometimes one can make it work just changing the initial guess.
- It fails if we are in a maximum or a minimum, $f'(x) = 0$.
- The evaluation of the derivative can be very expensive (using a finite-difference formula)
- It should be combined with a global method (like bisection).

Newton-Raphson method (one-dimension)

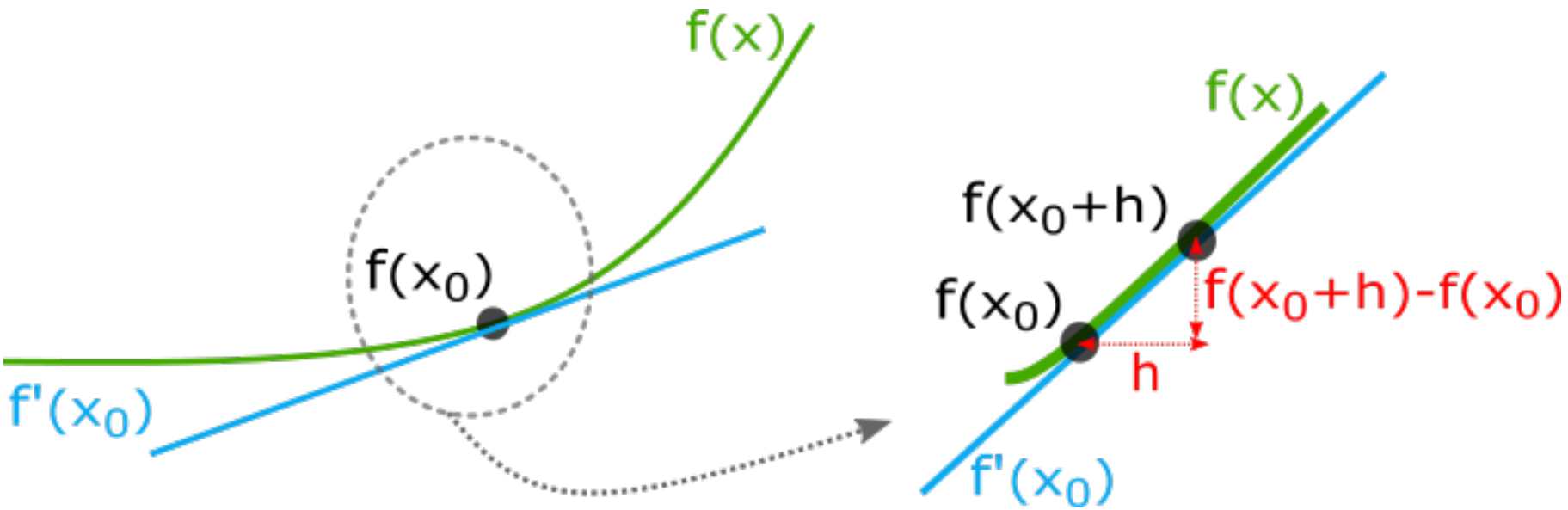
Algorithm:

1. Choose an initial point $x_n = x_0$.
2. Evaluate $f(x_n)$.
3. IF $f(x_n) \approx 0$ and x_n - $x_{n-1} \approx 0$ GO TO step 5.
4. ELSE ($f(x_n) \neq 0$) THEN
   - Compute $f^{'}(x_n)$
   - Compute the next point $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$
   - GO TO step 2
5. END program

Note: the user must provide a way to evaluate the function and the derivative.

How to evaluate first derivatives. Finite difference

Forward difference approximation: $\frac{df}{dx} = \frac{f(x_0+h)-f(x_0)}{h}$



Use a subroutine to calculate the gradient

How to evaluate first derivatives. Finite difference

Forward difference approximation: $\frac{df}{dx} = \frac{f(x_0+h)-f(x_0)}{h}$

Backward difference approximation: $\frac{df}{dx} = \frac{f(x_0)-f(x_0-h)}{h}$

Central difference approximation: $\frac{df}{dx} = \frac{f(x_0+h)-f(x_0-h)}{2h}$

Use a subroutine to calculate the gradient

Newton-Raphson method (one-dimension)

Algorithm:

1. Choose an initial point $x_n = x_0$.
2. Evaluate $f(x_n)$.
3. IF $f(x_n) \approx 0$ and x_n - $x_{n-1} \approx 0$ GO TO step 5.
4. ELSE ($f(x_n) \neq 0$) THEN
   - Compute $f^{'}(x_n)$
   - Compute the next point $x_{n+1} = x_n - \frac{f(x_n)}{f^{'}(x_n)}$
   - GO TO step 2
5. END program

Note: the user must provide a way to evaluate the function and the derivative.

How to make comparisons

- By definition: $f(root) = 0$
- But $\mathbf{x_{n+1} - x_n} \leq$ **tolerance** and $\mathbf{f(x_{n+1})} \leq$ **tolerance**.
- This is because computers use binary system. The result of a mathematical operation is not exactly the number we expect.
  - f(root)=$9.9999.10^{-15}$
  - so that f(root) .eq. 0 is False.
- Better to use a very small tolerance float number.
- Careful, **use absolute number for the comparison!**
  - **abs**$\mathbf{(x_{n+1} - x_n)} \leq$ tolerance
  - **abs**$\mathbf{(f(x_{n+1}))} \leq$ tolerance

Newton-Raphson method (multi-dimension)

Solving a system of n non-linear equations with n variables:

$$\begin{array}{l} f_1(x_1, x_2, x_3, \cdots x_n) = 0 \\ f_2(x_1, x_2, x_3, \cdots x_n) = 0 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ f_n(x_1, x_2, x_3, \cdots x_n) = 0 \end{array} \quad \Longrightarrow \quad F_i(x_1, x_2, x_3, \cdots x_n) = 0; \qquad (i = 1, \ldots, n)$$

or in matrix form $\mathbf{x} = \{x_i\}$ and $\mathbf{F} = \{F_i\}$

- Each of the functions can be approximated by a first order Taylor expansion in the neighborhood of $\mathbf{x}^k$

$$F_i(\mathbf{x}) = F_i(\mathbf{x}^k) + \sum_{j=1}^{n} \left(\frac{\partial F_i}{\partial x_j} \right)_{\mathbf{x}=\mathbf{x}^k} \left(x_j - x_j^k \right)$$

$$J_{ij}\text{(Jacobian Matrix)} \Rightarrow \frac{\partial F_i}{\partial x_j} = \begin{pmatrix} \partial f_1/\partial x_1 & \partial f_1/\partial x_2 & \cdots & \partial f_1/\partial x_N \\ \partial f_2/\partial x_1 & \partial f_2/\partial x_2 & \cdots & \partial f_2/\partial x_N \\ \vdots & \vdots & \ddots & \cdots \\ \partial f_N/\partial x_1 & \partial f_N/\partial x_2 & \cdots & \partial f_N/\partial x_N \end{pmatrix}$$

- $\mathbf{F}(\mathbf{x}) = \mathbf{F}(\mathbf{x}^k) + \mathbf{J} \cdot (\mathbf{x} - \mathbf{x}^k)$, with $\mathbf{J}$ evaluated at $\mathbf{x} = \mathbf{x}^k$
- Finding the roots ($\mathbf{F}(\mathbf{x}) = 0$) $\Rightarrow$

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \mathbf{J}^{-1} \cdot \mathbf{F}(\mathbf{x}^k) \qquad \text{iterative procedure}$$

Newton-Raphson method (multi-dimension)

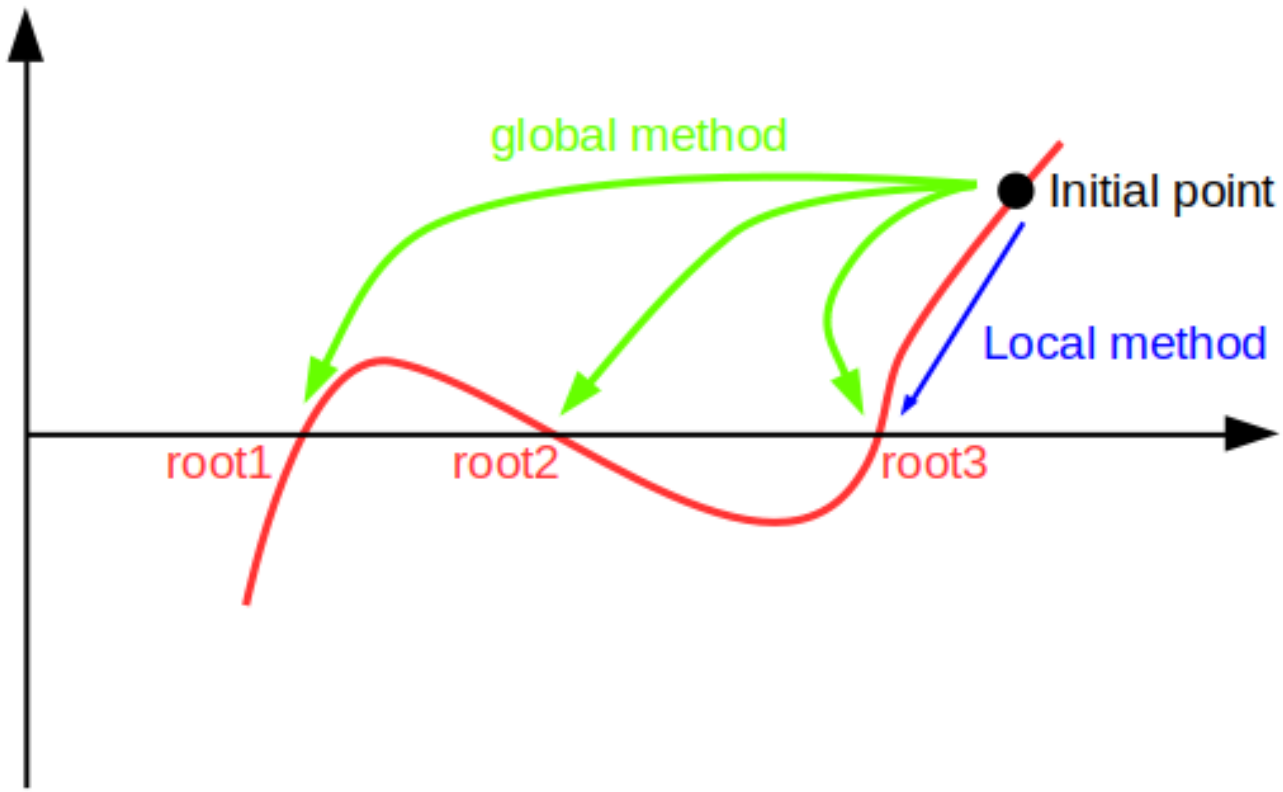
Algorithm:

1. Choose an initial point $\mathbf{x}^0$ (it is a N-dimensional vector).
2. Evaluate $\mathbf{F}(\mathbf{x}^k)$ (at the current point).
3. IF $\mathbf{F}(\mathbf{x}^k) \approx 0$ GO TO step 5.
4. ELSE ($\mathbf{F}(\mathbf{x}^k) \neq 0$) THEN
   - Compute the Jacobian matrix $\mathbf{J}$ evaluated at $\mathbf{x} = \mathbf{x}^k$.
   - Compute the $\mathbf{J}^{-1}$.
   - Calculate the next point $\mathbf{x}^{k+1} = \mathbf{x}^k - \mathbf{J}^{-1} f(\mathbf{x}^k)$
   - k = k + 1
   - GO TO step 2
5. END program

Summary of Roots of Functions:

- Successful root-finding combines global and local approaches. Global approaches assure that one stays in the region of interest. Local ones might quickly converge.
- Root finding in one dimension is straightforward. Once a root is bracketed, there are several methods to compute it (bisection, regula falsi, Newton-Raphson).
- Good initial points are very important.
- Root finding in higher dimensions is much more difficult. Global approach should be combined with local Newton-Raphson method.

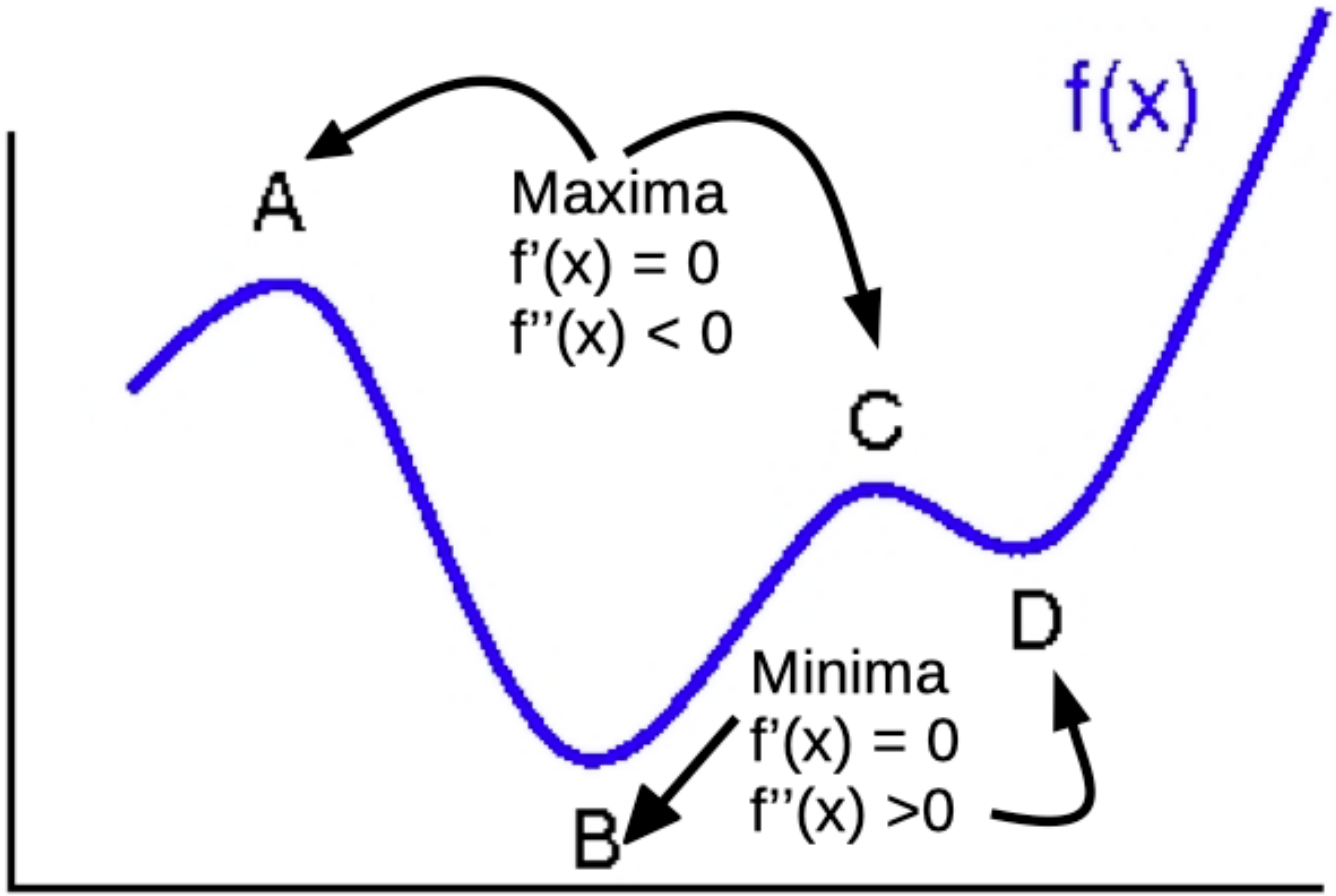
Local vs Global method



Outline

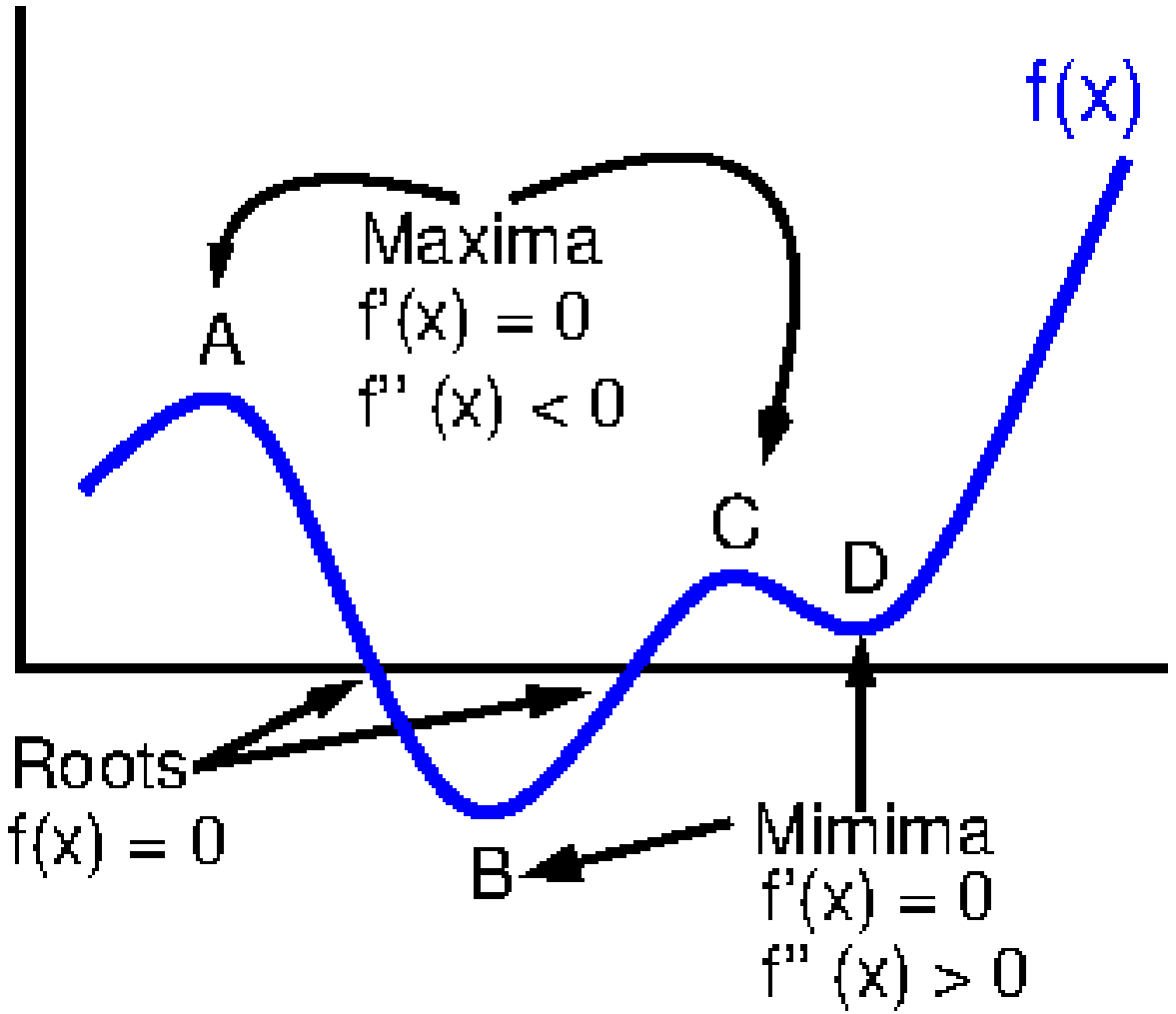
Function optimization

Is the simplest seeking the maxima and/or minima of some function relative to some set, representing a range of choices available in a certain situation. $\Rightarrow f^{'}(x_i) = 0$.



Function Optimization

Root-finding and function optimization are related problems



Function optimization: one-dimension

- Function optimization: one-dimensional
  1. Methods based on the function:
     - Interpolation methods: Parabolic interpolation.
  2. Methods based on the gradient of the function:
     - Newton-Raphson method.

Function optimization: one-dimension

1. Methods based on the function:
   - Interpolation methods: Parabolic interpolation.
     - Using equidistant points for the parabola
     - Using non-equidistant points for the parabola

There is only one possible parabola connecting three points.

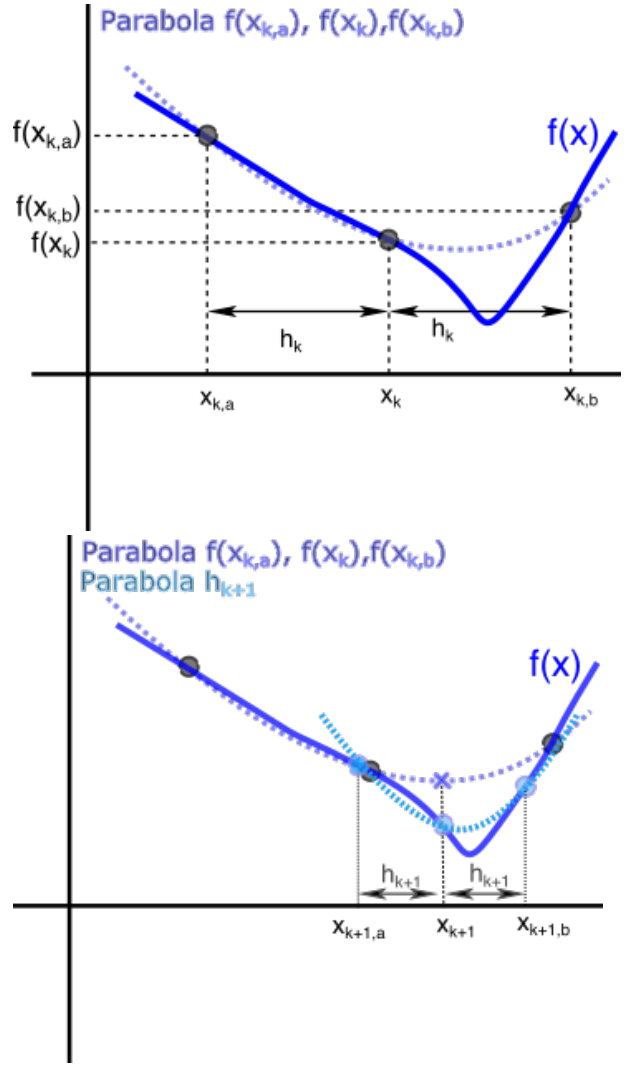
Using equidistant points for the parabola:

- Let's take a point $(x_k, f(x_k))$ of the function $f(x)$.
- Let's suppose that $f(x)$ has a minimum between $x_{k,a} = x_k - h_k$ and $x_{k,b} = x_k + h_k$.
- Evaluate $f_{k_a} = f(x_{k,a})$, $f_k = f(x_k)$ and $f_{k_b} = f(x_{k,b})$.
- The minimum of the parabola that goes through these points is at:

$$x_{k+1} = x_k - \frac{h_k}{2} \frac{f_{k_b} - f_{k_a}}{f_{k_b} - 2f_k + f_{k_a}}$$

- We repeat the process with a smaller step (h_k) until reaching convergence.

$$h_{k+1} = \frac{h_k}{a^k} \qquad \text{where } a = 2, 3.$$



Using equidistant points for the parabola:

Algorithm:

1. Choose an initial point $x_k = x_0$ and a step size h_k.
2. Evaluate $f(x_k)$, $f(x_{k+h_k})$ and $f(x_{k-h_k})$.
3. Compute the minima of the parabola
   $x_{k+1} = x_k - \frac{h_k}{2} \frac{f_{k_b} - f_{k_a}}{f_{k_b} - 2f_k + f_{k_a}}$
4. IF x_{k+1} - $x_k \approx 0$ GO TO step 6.
5. ELSE
   - Compute a smaller step $h_{k+1} = \frac{h_k}{a^k}$ where $a = 2, 3$
   - GO TO step 2
6. END program

Interpolation Methods: Parabolic or Quadratic

Using non-equidistant points for the parabola:

- Choose x_1, x_2, x_3 so that $f_2 < f_1$ and $f_2 < f_3$ being $f_1 = f(x_1)$, $f_2 = f(x_2)$ and $f_3 = f(x_3)$
- The minima of the parabola is at

$$x_4 = x_2 - \frac{1}{2}\frac{(x_2 - x_1)^2[f_2 - f_3] - (x_2 - x_3)^2[f_2 - f_1]}{(x_2 - x_1)[f_2 - f_3] - (x_2 - x_3)[f_2 - f_1]}$$

- Define $f_4 = f(x_4)$, and find the minima for the parabola that goes through x_1, x_4 and x_2
- Repeat this procedure until reaching convergence

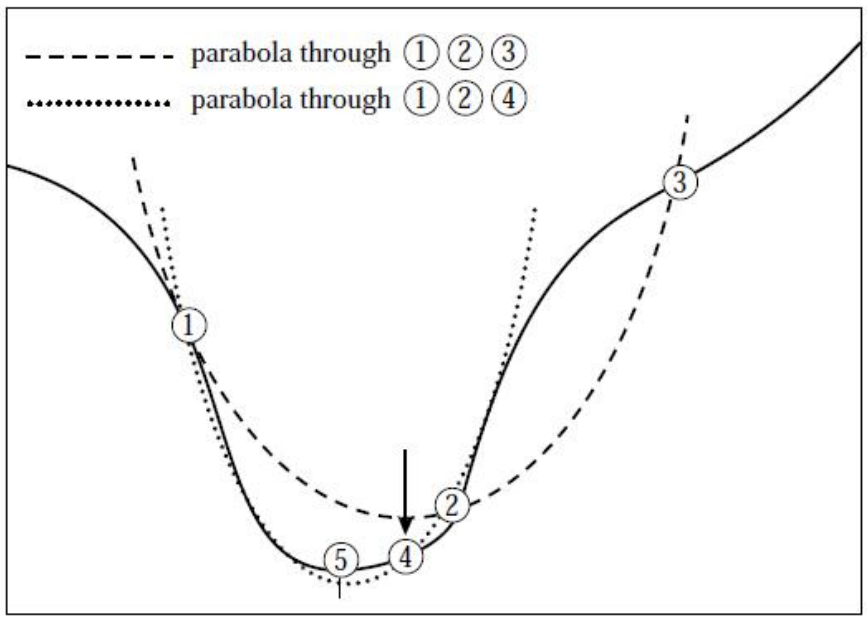


Figure 10.2.1. Convergence to a minimum by inverse parabolic interpolation. A parabola (dashed line) is drawn through the three original points 1,2,3 on the given function (solid line). The function is evaluated at the parabola's minimum, 4, which replaces point 3. A new parabola (dotted line) is drawn through points 1,4,2. The minimum of this parabola is at 5, which is close to the minimum of the function.

Interpolation Methods: Parabolic or Quadratic

Using non-equidistant points for the parabola:

Algorithm:

1. Choose three initial points x_1, x_2 and x_3
2. Evaluate $f(x_1)$, $f(x_2)$ and $f(x_3)$
3. IF $f(x_2) < f(x_1)$ and $f(x_2) < f(x_3)$ GO TO step 5
4. ELSE GO TO step 1
5. Compute the minima of the parabola
   $$x_4 = x_2 - \frac{1}{2}\frac{(x_2-x_1)^2[f_2-f_3]-(x_2-x_3)^2[f_2-f_1]}{(x_2-x_1)[f_2-f_3]-(x_2-x_3)[f_2-f_1]}$$
6. IF x_4 - $x_2 \approx 0$ GO TO step 8.
7. ELSE
   - Evaluate $f(x_4)$
   - GO TO step 5
8. END program

Function optimization: one-dimension

- Function optimization: one-dimensional
  1. Methods based on the function:
     - Interpolation methods: Parabolic interpolation.
  2. Methods based on the gradient of the function:
     - Newton-Raphson method.

Gradient method: Newton-Raphson

- In a minimum and a maximum: $f(x) \neq 0$ and $\mathbf{f'(x) = 0}$
- Apply the Newton-Raphson method for finding roots of the derivative $f'(x)$.
- $f(x)$ and $f'(x)$ are approximated at a given point x_0 by Taylor expansions (we ignore third and higher order derivatives):

$$\begin{aligned} \textit{Root finding} \Rightarrow f(x) &= f(x_0) + f'(x_0)(x - x_0) \\ \textbf{Optimization} \Rightarrow f'(x) &= f^{'}(x_0) + f''(x_0)(x - x_0) \end{aligned}$$

- If we are looking for a minimum/maximum: $f'(x) = 0$.

$$x_1 = x_0 - \frac{f^{'}(x_0)}{f''(x_0)}$$

- We repeat the process (iterative procedure) so that

$$x_{n+1} = x_n - \frac{f^{'}(x_n)}{f''(x_n)} \quad \& \quad f(x_{n+1}) < f(x_n)$$

until $x_{n+1} - x_n \leq$ tolerance and $f'(x) \approx 0$.

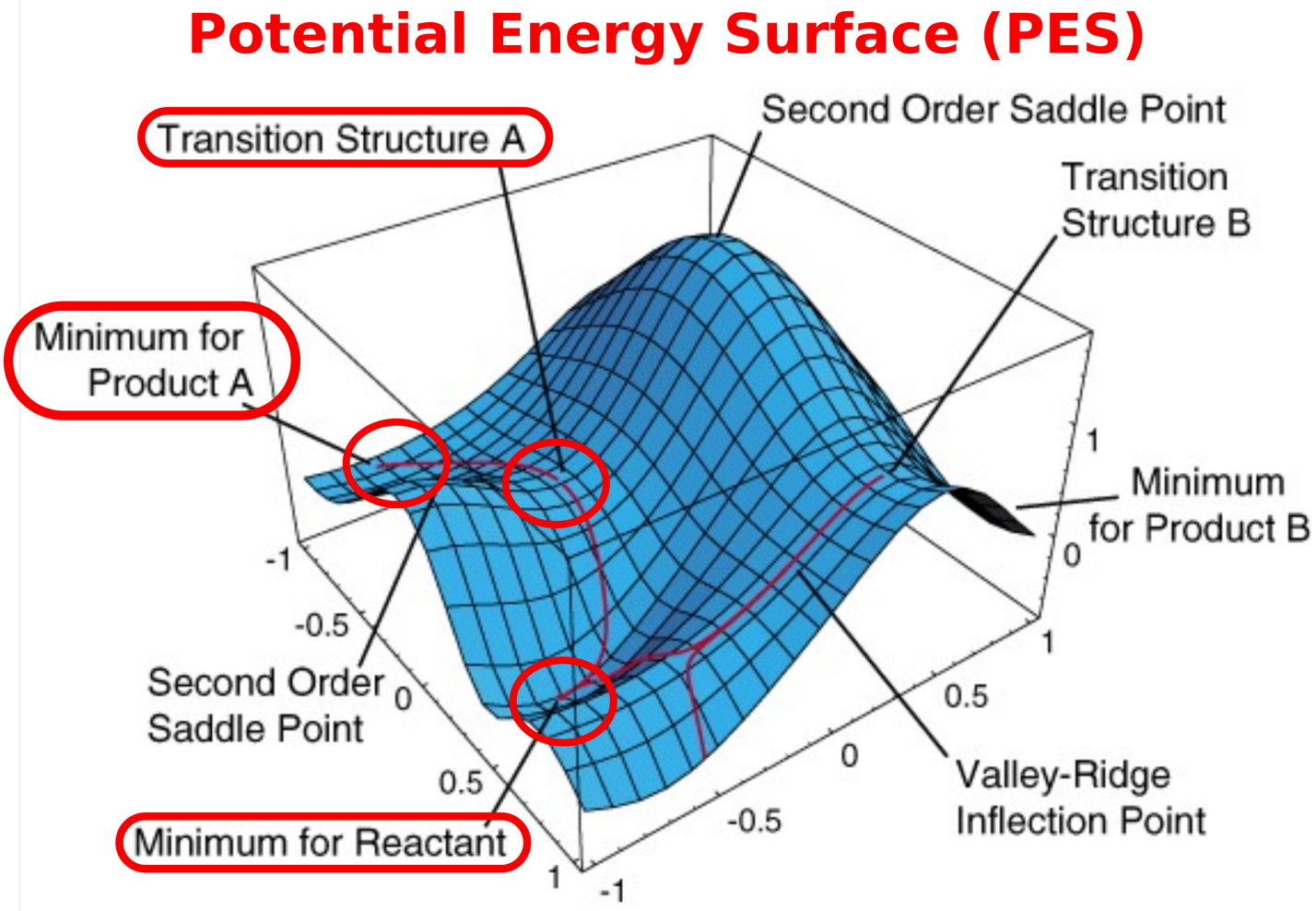
Gradient method: Newton-Raphson

Algorithm:

1. Choose an initial point $x_n = x_0$.
2. Evaluate $f'(x_n)$.
3. IF $f'(x_n) \approx 0$ and x_n - $x_{n-1} \approx 0$ GO TO step 5.
4. ELSE $(f'(x_n) \neq 0)$ THEN
   - Compute $f''(x_n)$
   - Compute the next point
     $x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$ & $f(x_{n+1}) < f(x_n)$
   - GO TO step 2
5. END program

Outline

Function optimization: multi-dimensional



Function optimization: multi-dimensional

General approach:

The main steps for most of optimization methods are:

1. choosing a starting point x_0.
2. choosing the direction d_k along which the next point is to be chosen.
3. deciding the step size λ_0 to be taken along that direction.

Then we have the following scheme:

$$x_{k+1} = x_k + \lambda_k d_k$$

Function optimization: multi-dimensional

Divided into three groups, depending on the magnitudes employed:

1. Methods based on the function, direct search methods:
   - Downhill Simplex method.
2. Methods based on the function and the gradient:
   - Steepest or gradient descent.
   - Conjugate gradient method.
3. Methods based on the gradient and the hessian:
   - Newton-Raphson method.

Note 1: Accuracy and computational time are very important in choosing a method.
Note 2: Better choose a method that makes use of the derivatives. Methods of category 3) will usually converge faster (in less number of steps), but it does not mean that they always are the most efficient. Computing and handling the second derivatives is hard, methods of category 2) could be a better choice.

Function optimization: multi-dimensional

They can be divided in three groups, depending on the magnitudes employed:

1. Methods based on the function, direct search methods:
   - Downhill Simplex method.
2. Methods based on the function and the gradient:
   - Steepest or gradient descent.
   - Conjugate gradient method.
3. Methods based on the gradient and the hessian:
   - Newton-Raphson method.

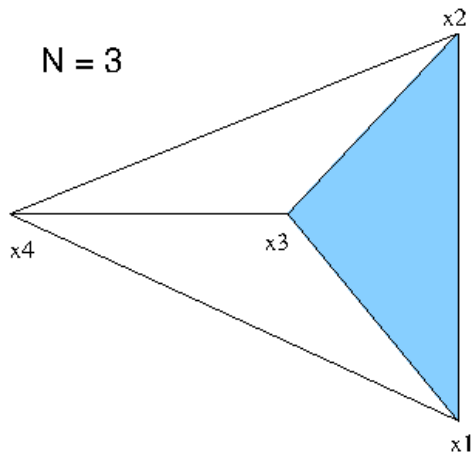
Note 1: Accuracy and computational time are very important in choosing a method.

Note 2: Better choose a method that makes use of the derivatives. Methods of category 3) will usually converge faster (in less number of steps), but it does not mean that they always are the most efficient. When computing and handling the second derivatives is hard, the methods of category 2) could be a better choice.

The downhill Simplex method

- Method due to Nelder and Mead. It requires function evaluations, not derivatives. (Do not confuse with Dantzig's simplex method of linear programming.)
- A **simplex** is defined as a geometrical figure of $N+1$ vertices in a space N-dimensional with no null hyper-volume

- $N = 2$ (plane)$\Longrightarrow$Triangle
- $N = 3 \Longrightarrow$Tetrahedrom
- .....



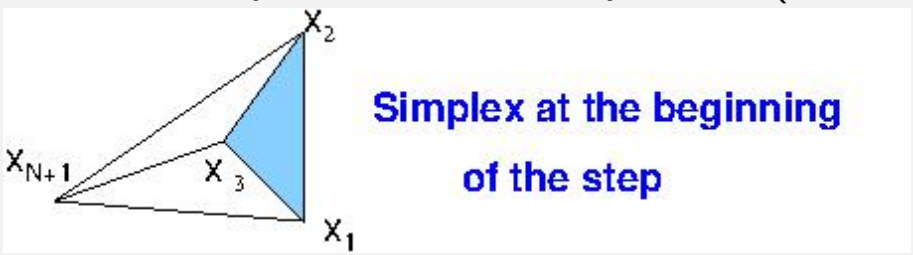
If any vertex of a simplex is taken as the origin ($\mathbf{P}_0$), then the other N points define vectors directions that span the N-dimensional space.

$$\mathbf{P}_i = \mathbf{P}_0 + \lambda \mathbf{e}_i \quad (i = 1, \cdots, N)$$

where $\mathbf{e}_i$ are the unitary vectors.

Algorithm:

1. Definition of the inital simplex with $N+1$ points: $(x_1,\ x_2,\ \cdots\ x_{N+1})$.



2. Evaluation of the function at the $N+1$ points, and sorting them in increasing order: $f(x_1) \leq f(x_2) \leq \cdots \leq f(x_{N+1})$
3. IF standard deviation of the function at N+1 points $\leq$ converge
    1. STOP
    2. ELSE GO TO Step 4

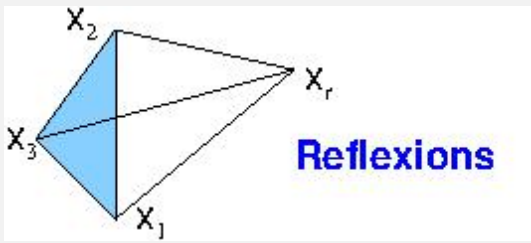
4. Computation of x_0 as the center of all vertices except x_{N+1}

$$x_0 = \frac{1}{N}\sum_{j=1}^{N} x_j$$

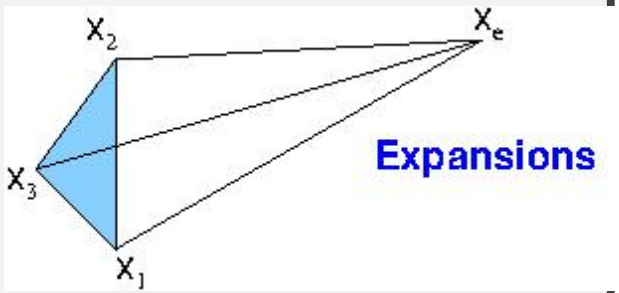
5. Reflexion respect to x_0: calculate the reflexion point

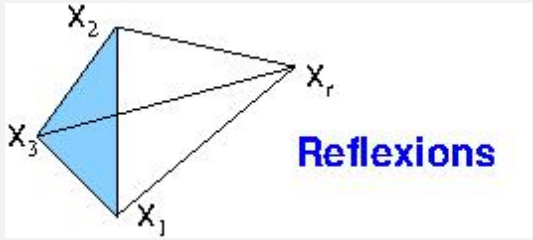
$$x_r = x_0 + \alpha(x_0 - x_{N+1}); \qquad \alpha = 1$$

6. **IF $\mathbf{f(x_1) \leq f(x_r) \leq f(x_N)}$**
   1. Accept x_r as new vertex
   2. Eliminate the worst vertex, x_{N+1}
   3. GO TO step 2

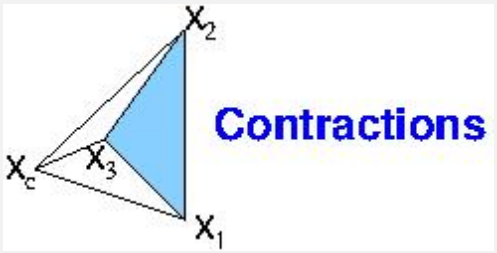


6. **ELSE IF $f(x_r) < f(x_1)$**
   1. Expansion: calculate the expansion point
      $$x_e = x_0 + \alpha\delta(x_0 - x_{N+1}); \qquad \alpha = 1,\ \delta = 2$$
   2. Evaluate $f(x_e)$
   3. **IF $f(x_e) < f(x_r)$**
      1. Accept x_e as new vertex
      2. Eliminate the worst vertex, x_{N+1}
      3. GO TO step 2
   4. **ELSE IF $f(x_e) \geq f(x_r)$**
      1. Accept x_r as new vertex
      2. Eliminate the worst vertex, x_{N+1}
      3. GO TO step 2



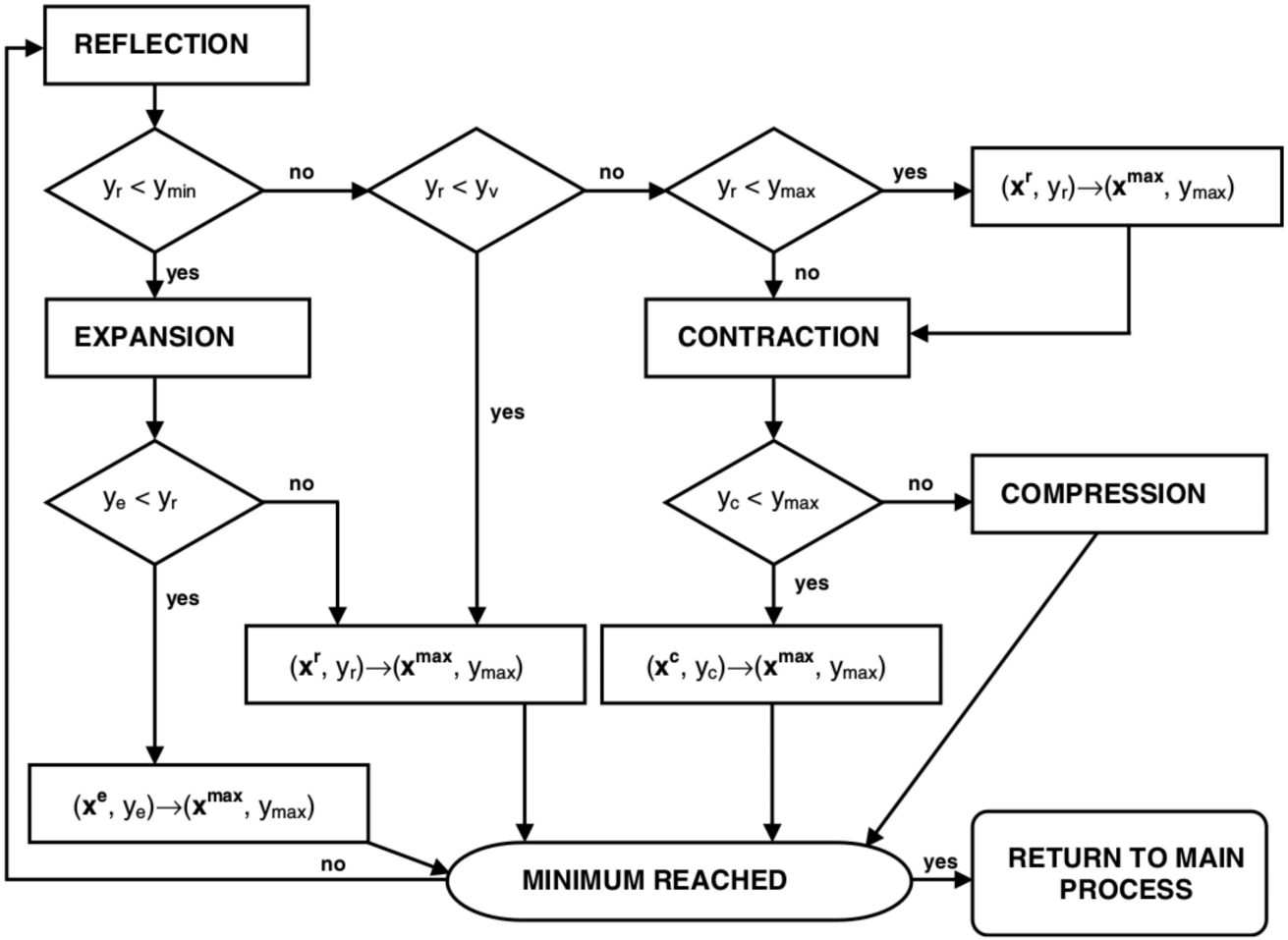


7. **ELSE IF $f(x_N) \leq f(x_r) < f(x_{N+1})$**
   1. Contraction: calculate the external contraction point
      $$x_c = x_0 + \alpha\gamma(x_0 - x_{N+1}); \qquad \alpha = 1,\ \delta = 2,\ \gamma = 1/2$$
   2. Evaluate $f(x_c)$
   3. **IF $f(x_c) \leq f(x_r)$**
      1. Accept x_c as new vertex
      2. Eliminate the worst vertex, x_{N+1}
      3. GO TO step 2
   4. **ELSE IF $f(x_c) > f(x_r)$**
      1. GO TO step 9



8. **ELSE IF $f(x_r) \geq f(x_{N+1})$**
   1. Contraction: calculate the internal contraction point
      $$x_c = x_0 - \gamma(x_0 - x_{N+1}); \qquad \alpha = 1,\ \delta = 2,\ \gamma = 1/2$$
   2. Evaluate $f(x_c)$
   3. **IF $f(x_c) \leq f(x_{N+1})$**
      1. Accept x_c as new vertex
      2. Eliminate the worst vertex, x_{N+1}
      3. GO TO step 2
   4. **IF $f(x_c) > f(x_{N+1})$**
      1. GO TO step 9
9. Shrink: Replace all vertices except the best one, x_1
   $$x_i = x_1 + \sigma(x_i - x_1); \quad i = 2, ..., N+1; \quad \sigma = 1/2$$
   1. GO TO step 2

The downhill simplex method



It can converge to a non-stationary point.

https://commons.wikimedia.org/wiki/File:Nelder-Mead_Himmelblau.gif#/media/File:Nelder-Mead_Himmelblau.gif

Function optimization: multi-dimensional

They can be divided in three groups, depending on the magnitudes employed:

1. Methods based on the function, direct search methods:
   - Downhill Simplex method.
2. Methods based on the function and the gradient:
   - Steepest or gradient descent.
   - Conjugate gradient method.
3. Methods based on the gradient and the hessian:
   - Newton-Raphson method.

Note 1: Accuracy and computational time are very important in choosing a method.

Note 2: Better choose a method that makes use of the derivatives. Methods of category 3) will usually converge faster (in less number of steps), but it does not mean that they always are the most efficient. When computing and handling the second derivatives is hard, the methods of category 2) could be a better choice.

Steepest or gradient descent method

Steepest descent method

- It is the simplest iterative method that uses the information of the gradient to find the minimum of a function.
- It may be applied to any function in which the gradient can be computed.

Algorithm

1. Choose an initial starting point $\mathbf{P}_0 = (x_1^0, x_2^0, \cdots, x_N^0)$.
2. Computation of steepest direction of $f(\mathbf{P})$. The steepest direction is obtained as the negative gradient of f at $\mathbf{P}_0$, that is, in the opposite direction of the gradient.
3. Calculation of the gradient at $\mathbf{P}_0$, $\nabla f(\mathbf{P}_0)$.
4. Evaluate next point
$$\mathbf{P}_{k+1} = \mathbf{P}_k - \gamma_k \nabla f(\mathbf{P}_k)$$
5. Repeat this successively until convergence, $\mathbf{P}_{k+1} - \mathbf{P}_k \leq$ tolerance

Steepest or gradient descent method

Which should be the step size?

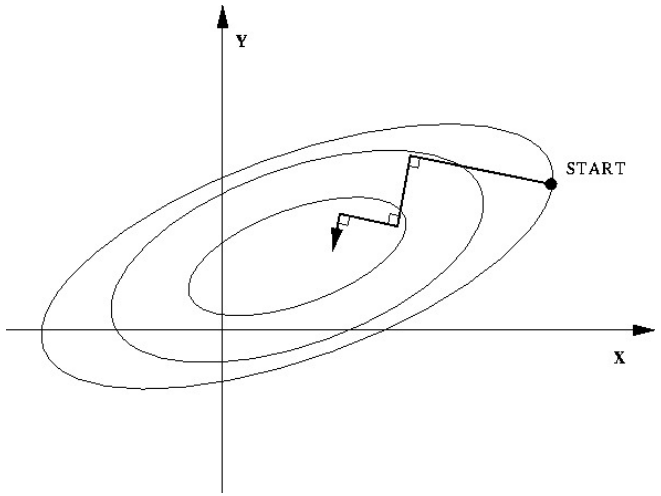
- the one that leads the minimization to a point where the function takes on a minimum value:

$$\frac{d}{d\gamma_k} f(\mathbf{P}_{k+1}) = 0 \qquad \mathbf{P}_{k+1} = \mathbf{P}_k - \gamma_k \nabla f(\mathbf{P}_k)$$

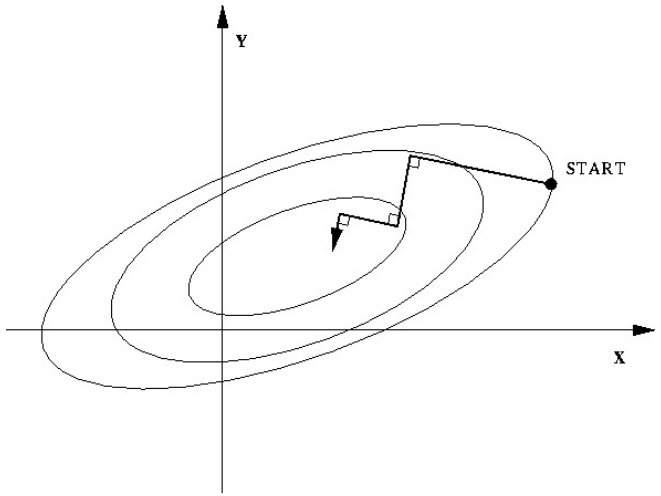
$$\frac{d}{d\gamma_k} f(\mathbf{P}_{k+1}) = \frac{df(P_{k+1})}{d(P_{k+1})} \cdot \frac{d(P_{k+1})}{d\gamma_k} = \nabla f(\mathbf{P}_{k+1}) \cdot \{-\nabla f(\mathbf{P}_k)\}$$

$$0 = -\nabla f(\mathbf{P}_{k+1}) \cdot \nabla f(\mathbf{P}_k)$$

- $\nabla f(\mathbf{P}_{k+1}) \cdot \nabla f(\mathbf{P}_k)$ are orthogonal $\rightarrow$ characteristic zig-zag path.
- However, we will use a fixed step size. Simpler, but less efficient.



Steepest or gradient descent



Advantages:

- Simple. Minimization at each step is guaranteed.
- Fast minimization far from the minimum.

Disadvantages:

- Slow descent along narrow valleys. Slow convergence near minima.
- Real stationary point not reached (second derivative needed).
- Useful to approach the minimum, and then another optimization method (conjugate gradient) is recommended.

Function optimization: multi-dimensional

They can be divided in three groups, depending on the magnitudes employed:

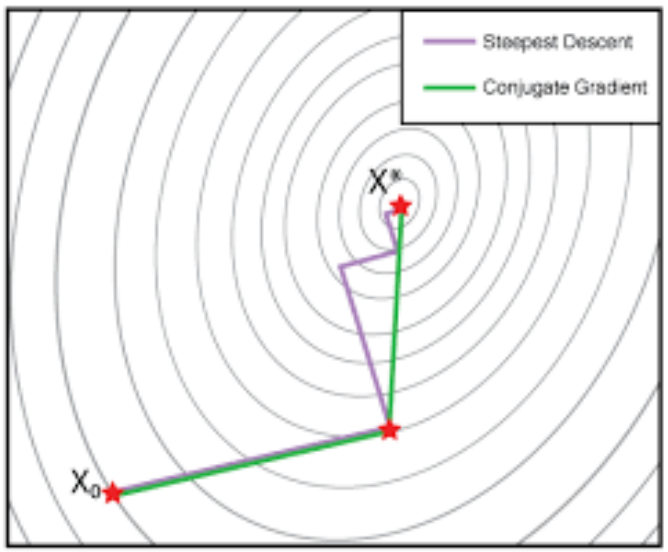
1. Methods based on the function, direct search methods:
   - Downhill Simplex method.
2. Methods based on the function and the gradient:
   - Steepest or gradient descent.
   - Conjugate gradient method.
3. Methods based on the gradient and the hessian:
   - Newton-Raphson method.

Note 1: Accuracy and computational time are very important in choosing a method.
Note 2: Better choose a method that makes use of the derivatives. Methods of category 3) will usually converge faster (in less number of steps), but it does not mean that they always are the most efficient. When computing and handling the second derivatives is hard, the methods of category 2) could be a better choice.

Conjugated Gradient Method

1. Initialized by a steepest descent step.
2. Subsequent steps follow a line formed as a mixture of the current negative gradient and the previous search direction (it is conjugated):

$$d_i = -g_i + \beta\, d_{i-1} = -\nabla f(\mathbf{P}_i) - \beta\, \nabla f(\mathbf{P}_{i-1})$$



Alternative β values. For instance, Polak-Ribiere (PR):

$$\beta_i^{PR} = \frac{g_i^T (g_i - g_{i-1})}{g_{i-1}^T g_{i-1}}$$

- Fast minimization near the minima

Function optimization: multi-dimensional

They can be divided in three groups, depending on the magnitudes employed:

1. Methods based on the function, direct search methods:
   - Downhill Simplex method.
2. Methods based on the function and the gradient:
   - Steepest or gradient descent.
   - Conjugate gradient method.
3. Methods based on the gradient and the hessian:
   - Newton-Raphson method.

Note 1: Accuracy and computational time are very important in choosing a method.

Note 2: Better choose a method that makes use of the derivatives. Methods of category 3) will usually converge faster (in less number of steps), but it does not mean that they always are the most efficient. When computing and handling the second derivatives is hard, the methods of category 2) could be a better choice.

Newton-Raphson method

- This method uses the information of the second derivative to locate the minimum of f in an N-dimensional problem:
- In each iteration f is approximated by a quadratic function (2nd order Taylor expansion) around a chosen point $\mathbf{x}^k$, and move to the minimum of this function.

$$f(\mathbf{x}) = f(\mathbf{x}^k) + \nabla f(\mathbf{x}^k)(\mathbf{x} - \mathbf{x}^k) + \frac{1}{2}\mathbf{H}(\mathbf{x}^k)(\mathbf{x} - \mathbf{x}^k)^2$$

This implies

$$\nabla f(\mathbf{x}) = \nabla f(\mathbf{x}^k) + \mathbf{H}(\mathbf{x}^k)(\mathbf{x} - \mathbf{x}^k)$$

$$\mathbf{H}(\mathbf{x}^k) = \left(\begin{array}{ccc} \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_2} & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_3} \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \partial x_2} & \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \partial x_3} \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_3 \partial x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_3 \partial x_2} & \frac{\partial^2 f(\mathbf{x})}{\partial x_3 \partial x_3} \end{array} \right)_{\mathbf{x}=\mathbf{x}^k} \quad \textbf{(Hessian)}$$

- Condition of a minimum: $\nabla f(\mathbf{x}) = 0$

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \left[H(\mathbf{x}^k)\right]^{-1} \cdot \nabla f(\mathbf{x}^k)$$

where $-\mathbf{H}(\mathbf{x}^k)^{-1} \cdot \nabla f(\mathbf{x}^k)$ indicates the direction.

Algorithm

1. Consider an error tolerance, ϵ, for $\nabla f(\mathbf{x})$ and $(\mathbf{x}^{k+1} - \mathbf{x}^k)$.
2. Take a starting point $\mathbf{x}^k$ (N-dimensional vector)
3. Evaluate the gradient $\nabla f(\mathbf{x}^k)$ (gradient calculus subroutine)
4. IF $\nabla f(\mathbf{x}^k) > \epsilon$ THEN
   - Compute Hessian (Hessian, inversion and matrices multiplication subroutines)
   - Calculate next point: $\mathbf{x}^{k+1} = \mathbf{x}^k - \left[H(\mathbf{x}^k)\right]^{-1} \nabla f(\mathbf{x}^k)$
   - GO TO step 3
5. ELSE GO TO step 6
6. END program

Newton-Raphson method

Some important remarks:

- Newton-Raphson method converges faster than the steepest descent and conjugate gradiente methods, because it uses the information of the curvature of the function given by the Hessian.
- The method is very dependent on the initial point. It converges faster when closer to the minimum, in contrast to the steepest descent. They can be combined.
- When finding a minimum, the Hessian should be always positive definiteness (there are some methods to assure this).
- Computing and handling the Hessian can be very hard.
- The size of the Hessian can be a problem specially for large systems. Several algorithms have been developed in order to avoid recalculating the Hessian at each step and storing it: **quasi-Newton methods**.

Quasi-Newton methods

- Avoid recalculating the Hessian at each step.
- The Hessian is estimated using the information from previous steps.
  - Initial Hessian matrix is either empirical or computed at a lower level of theory.
  - It is subsequently updated by a formula:

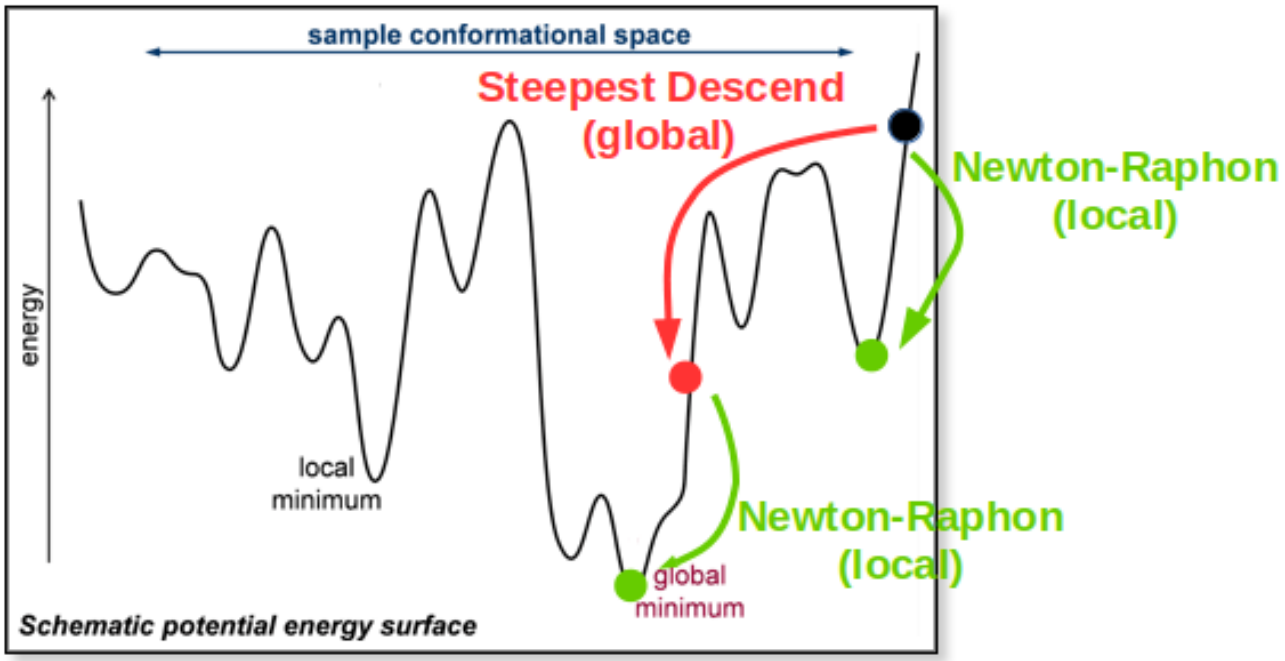
$$H^{new} = H^{old} + \triangle H$$

- For example, the **BFGS** (Broyden-Fletcher-Goldfarb-Shanno) updating formulas:

$$\triangle H_{BFGS} = \frac{\triangle g \triangle g^T}{\triangle g^T \triangle x} - \frac{H^{old} \triangle x \triangle x^T H^{old}}{\triangle x^T H^{old} \triangle x}$$

- Thus, vectors ($\triangle x$ and $\triangle g$) are used to estimate the Hessian matrix.
- The **Berny** updating algorithm is widely used in the GAUSSIAN program.

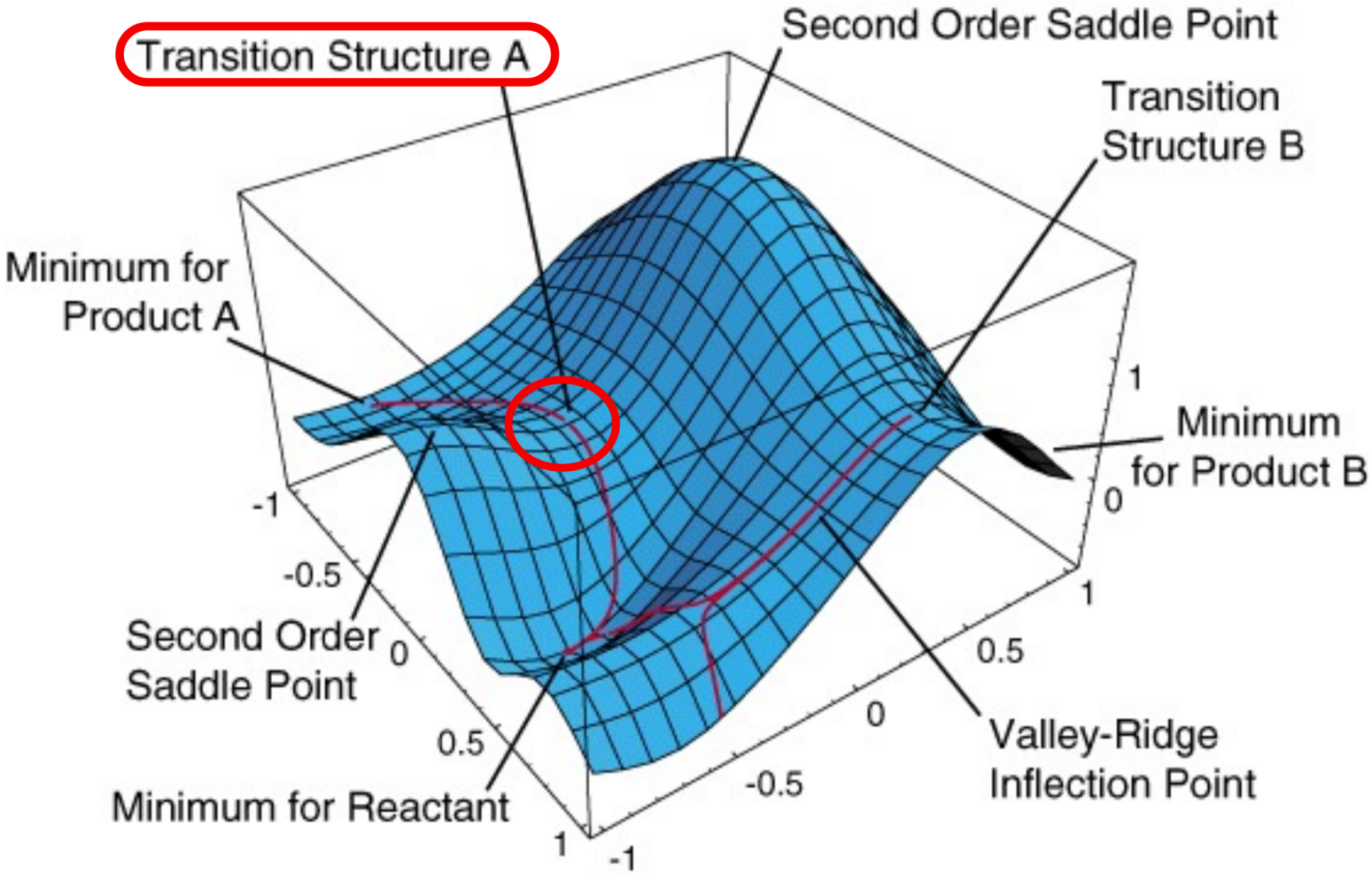
Local vs Global Optimizations



Recommendation: global method (Steepest descent) first, followed by local method (Newton-Raphson)

Outline

Potential Energy Surface (PES)



Locating transition states on PES

Transition states (TS)

- Specially important when studying chemical reactions.
- Searching for first order saddle points is more difficult than finding a minimum.
- The saddle point must fulfill some "chemical conditions":
  - A TS is the highest-energy point along a continuous line connecting reactants and products.
  - If there are more than one, the TS is energetically the lowest one.

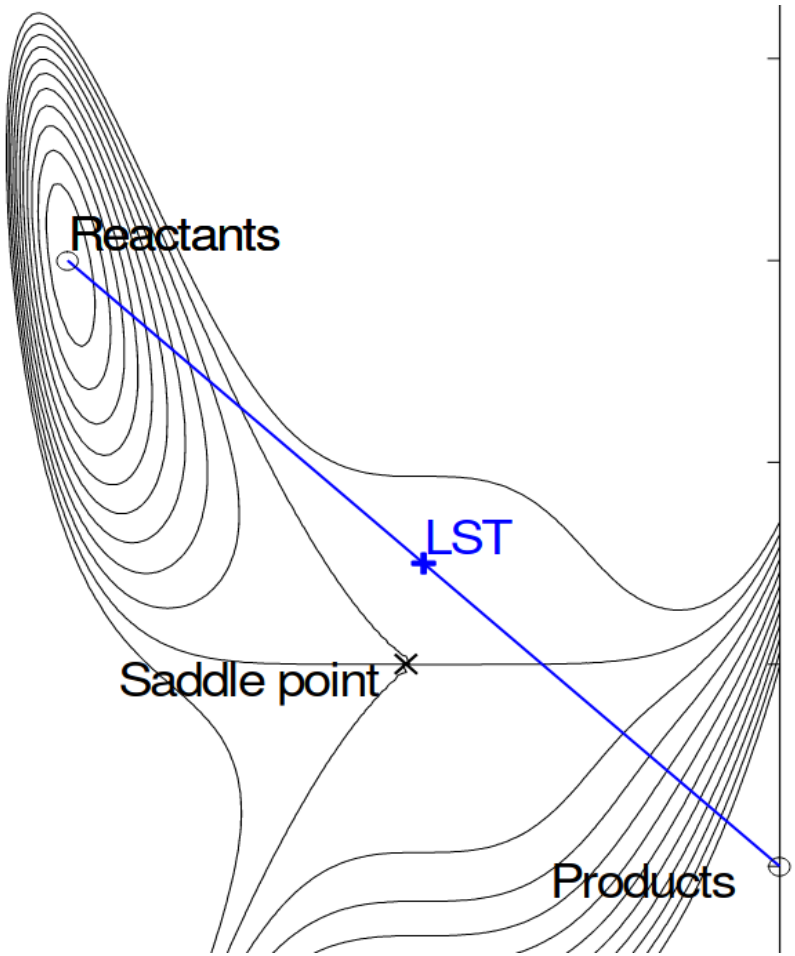
Algorithms for finding transition states

1. Global methods: interpolation between reactants and products.
   - Linear Synchronous Transit (LST)
   - Quadratic Synchronous Transit (QST)
2. Local methods: augmented Newton-Raphson
   - Eigenvector following method (EF)
3. Synchronous Transit-Guided Quasi-Newton (STQN) method: QST + quasi-Newton

Algorithms for finding transition states

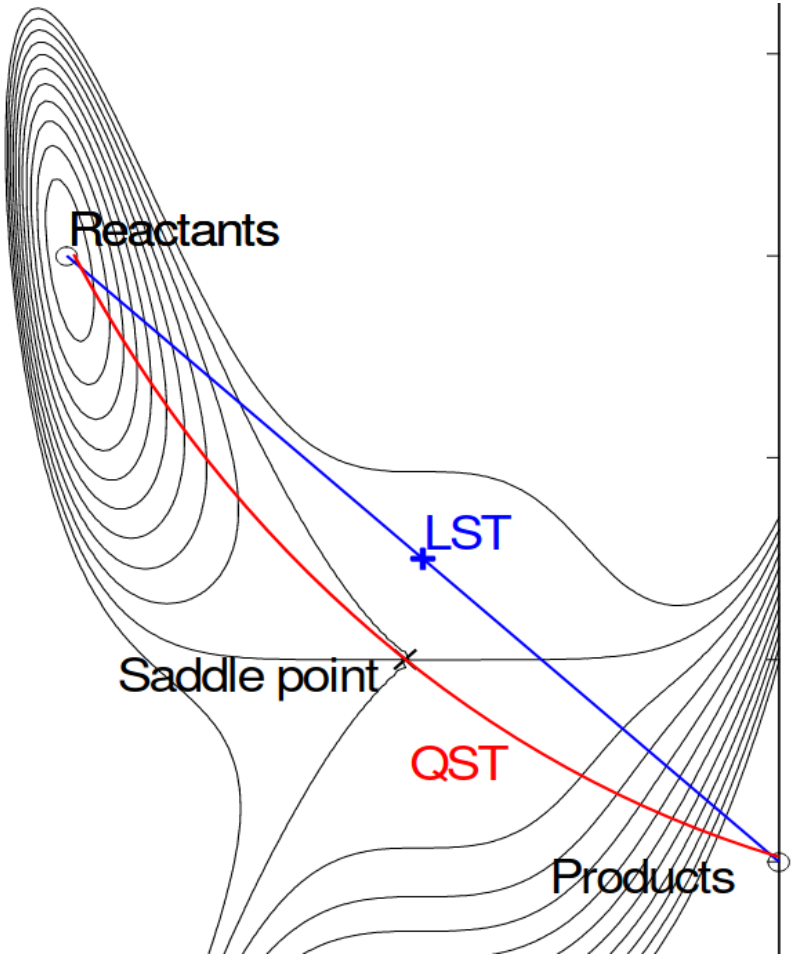
1. **Global methods**: interpolation between reactants and products.
   - Linear Synchronous Transit (LST)
   - Quadratic Synchronous Transit (QST)
2. Local methods: augmented Newton-Raphson
   - Eigenvector following method (EF)
3. Synchronous Transit-Guided Quasi-Newton (STQN) method: QST + quasi-Newton

Linear Synchronous Transit Methods (LST):



- The Linear synchronous transit methods (LST) searches for the energy maximum on the linear path connecting reactants and products.
- The initial guess will be a structure exactly halfway between the reactants and products.
- All degrees of freedom are varied linearly between reactant and product.
- Assumptions:
  - All variables change at the same rate along the reaction path.
  - Transition state is the highest energy structure along the interpolation line.
- LST is a poor approximation. This method rarely leads to a good estimate of the TS.

Quadratic Synchronous Transit Methods (QST):



- Quadratic synchronous transit methods (QST) is an improvement on LST.
- QST interpolates the path between reactants and products using a parabola.
  1. Find the maximum on the LST.
  2. Generate QST by minimizing energy perpendicular to the LST path.
  3. Search for maximum on QST path.
- It provides an **approximate location of the TS** that can be used as a starting point for other methods (STQN).

Algorithms for finding transition states

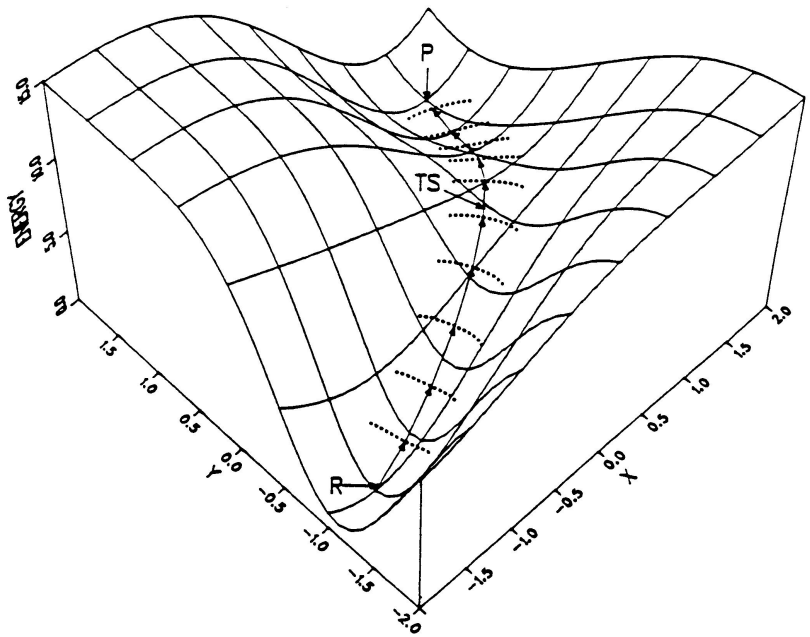
1. Global methods: interpolation between reactants and products.
   - Linear Synchronous Transit (LST)
   - Quadratic Synchronous Transit (QST)
2. **Local methods**: augmented Newton-Raphson
   - Eigenvector following method (EF)
3. Synchronous Transit-Guided Quasi-Newton (STQN) method: QST + quasi-Newton

Locating transition states on PES: **EF method**

Eigenvector-following (EF) method:

The eigenvector-following technique can find transition states on a PES by **maximizing the energy along a chosen eigenvector of the Hessian while simultaneously minimizing the energy along all other eigenvectors.**

1. At the initial minimum, compute and diagonalize the Hessian.
2. Move in the direction of the eigenvector of interest (lowest one) by a user-specified intial step.
3. Compute the gradient and the Hessian at the new point. Diagonalize the Hessian.
4. Take Newton steps in all directions except the one of the lowest eigenvalue. Go uphill in that direction.
5. Repeat from 2. until convergence.



Advantages: very robust.
Disadvantages: very expensive (a Hessian is computed at each step). No guarantee that the saddle point is the one wanted.

Algorithms for finding transition states

1. Global methods: interpolation between reactants and products.
   - Linear Synchronous Transit (LST)
   - Quadratic Synchronous Transit (QST)
2. Local methods: augmented Newton-Raphson
   - Eigenvector following method (EF)
3. **Synchronous Transit-Guided Quasi-Newton (STQN) method**: QST + quasi-Newton

Synchronous Transit-Guided Quasi-Newton (STQN) methods:

- this method implemented by H. B. Schlegel and coworkers uses a quadratic synchronous transit approach (QST) to get closer to the quadratic region of the transition state and then uses a quasi-Newton or eigenvector-following algorithm to complete the optimization.

Some Remarks

Interpolation methods:

- Reactant and product minima must be known.
- A TS is located "between" these two end-points.
- May only find a geometry close to the TS, rather than the true TS.

Local methods:

- Propagate geometry using information about: i) the function, ii) first and iii) second derivatives.
- Do not need to know reactant/product.
- Usually need to have a good estimate of the TS.

Optimization methods in the available software

- **MOLPRO program**
  - Minima and TS optimization: GDIIS, QSD
  - Update Method: BFGS (default), Powell, numerically, ..
- **COLUMBUS program**
  - Minima and TS optimization: GDIIS, ...
  - Update Method: Powell, BFGS ...
- **GAUSSIAN program: (Opt)**
  - Geometry optimization: Berny+GEDIIS (default)
  - Location of TS: STQN (QST2 and QST3), Berny+GEDIIS (default)
  - Update Method: Powell, BFGS ...
- **GAMESS program**
  - Minima and TS optimization: NR, QNR, Monte-Carlo, ...
  - Update Method: Powell, BFGS ...
- **ACESII program**
  - Minima and TS optimization: analytically evaluated gradients within a QN scheme (default)
  - Update Method: BFGS ...
  - If a Hessian matrix is available, a Newton-Raphson scheme is applied.

References


A. R. Krommer and C. W. Ueberhuber
Numerical Integration on Advanced Computer Systems
(Springer-Verlag Berlin, Heidelberg, 1994)


P. J. Davis and P. Rabinowitz
Methods of Numerical Integration (second edition, Academic Press, Inc., London, 1984)


S. C. Chapra and R. C. Canale
Numerical Methods for engineers (fifth edition, The McGraw-Hill Companies, Inc., México, 2006)


C. A. Floudas and P. M. Pardalos
Optimization in Computational Chemistry and Molecular Biology: Local and Global Approaches (1st edition, Springer, 2000)


S. A. Teukolsky, W. T. Vetterling and B. P. Flannery
Numerical Recipes in Fortran 77 (second edition, Univ. Press, Cambridge, 2003)


J. Andrés and J. Bertran eds.
Theoretical and Computational Chemistry (Universitat Jaume I publications, Castellón de la Plana, 2007)

Homework

Deadlines

- 10th December: Numerical Integration homework (continuous evaluation, no marks)
- 17th December: Root-finding and function optimization homework (continuous evaluation, no marks)
- 10th March: Evaluation homeworks

How to write the programs

A good code:

- It must provide the right answer ... and more:
  - It must be compiled without errors.
  - It must be well-structured and properly explained with comments. Add comments!
  - It produces a concise but understandable output.
  - It contains sub-programs (functions and subroutines):
    - Use functions to calculate the values of a f(x) function.

All these factors will be evaluated.