# Introduction, basic but important concepts

Felix Kubler[1]

[1] DBF, University of Zurich and Swiss Finance Institute

June 19, 2019

# Economics or Math

- Numerical analysis and scientific computing light-years ahead of us
- Try to focus on a set of economic problems and search for best methods available for this problem

# This week

- Standard stuff: Non-linear equations, optimization.
- Somewhat standard stuff: Numerical dynamic programming, projection methods
- Less standard stuff: Deep reinforcement learning

# This week

1. Introduction, Non-linear equations, Non-linear programming
2. Basic Dynamic programming
3. Integration and function approximation
4. Smooth dynamic programming, Projection Methods
5. Reinforcement learning

# Errors

In general there are 2 sources of errors in numerical computations: rounding and truncation errors.

- Rounding errors are a consequence of using finite precision arithmetic.
- Truncation errors: Even if one has *convergent* algorithm, for which

$$\lim_{n\to\infty} F_n(x_n, d) \to F(x, d),$$

one only has finite time and has to stop at some $N < \infty$. Any algorithm we use will therefore stop at some approximation $\hat{x}$ to a $x^*$ with $F(x^*, d) = 0$. In general, we would be happy with a tiny relative error, i.e. with

$$\frac{|x^* - \hat{x}|}{x^*} \approx u.$$

This is called a small *forward* error. However, in some circumstances all we can hope for is a small *backward error*, i.e. that $F(\hat{x}, \hat{d}) = 0$ for some $\hat{d}$ with $\hat{d} - d \equiv u$.

# Finite precision arithmetics - examples

- Consider the function

$$f(x) = (1 - \cos(x))/x^2$$

with $x = 1.2 \times 10^{-5}$ the value of cos, rounded to 10 significant digits is $c = 0.9999999999$ so that

$$1 - \cos(x) = 0.00000001$$

Then $(1 - c)/x^2 = 10^{-10}/1.44 \times 10^{-10} = 0.69...$ Unfortunately, the correct result is 0.5

- Suppose we are trying to solve the quadratic equation

$$10^{20}x^2 - 3 \cdot 10^{20}x + 2 \cdot 10^{20} = 0$$

Using the simple formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a},$$

the result will almost certainly be *NaN*.

# Linear algebra

- Numerical linear algebra is an extremely well developed field
- Excellent code in Fortran, C, and Matlab were developed in the 1970's and 1980's.
- One prominent problem is t solve a linear system:
  Solve $Ax = b$ for a vector $x \in \mathbb{R}^n$, assuming that the $n \times n$ matrix $A$ is an invertible.

# Direct methods to solve linear system

NEVER use $x = A^{-1}b$. Instead:

- $LL'$ (Cholesky) factorization (sparse, symmetric, positive definite)
- $LDL'$ factorization (sparse, symmetric, indefinite)
- $LU$ factorization (sparse, generic, full rank)
- $QR$ factorization (dense, generic)
- $USV'$ singular value decomposition (dense, generic)

# Iterative methods

One guesses a starting $x^0$ then iterates on

$$x^{k+1} = Bx^k + f$$

until $\|x^{k+1} - x^k\|$ becomes small. Hopefully $B$ and $f$ can be chosen to ensure that $\|x^k - x^*\| \to 0$ for $x^*$ with $Ax^* = b$.

Two important methods: Gauss-Seidel and Gauss-Jacobi

# Gauss-Jacobi

For each $i = 1, ..., n$, set

$$x_i^{k+1} = \frac{1}{a_{ii}} \left( b_i - \sum_{j=1, j \neq i}^{n} a_{ij} x_j^k \right)$$

Note that, in this scheme $B = I - D^{-1}A$ where $D$ is a diagonal matrix with the diagonal elements of $A$ on its diagonal, $f = D^{-1}b$. Note that, fortunately, $x = Bx + f$ and $Ax = b$ are equivalent. Therefore, if $x \approx Bx + f$ that then $Ax \approx b$, so if the iterations converge they converge to the right solution.

# Gauss-Seidel

Sometime it is useful to use information from iteration $k+1$ already in that iteration. One example is Gauss-Seidel

$$x_i^{k+1} = \frac{1}{a_{ii}} \left( b_i \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k \right)$$

Lower storage requirements. Perhaps faster.

# Iterative methods (concluded)

- Methods are easy to code and can solve huge systems
- Convergence only guaranteed under unrealistic assumptions.
- Can be very slow, but we can use acceleration methods. For example Chebychev second-order iterative schemes. Here we set

$$\alpha_{i+1} = \omega G(\alpha_i) + (\tau - \omega)\alpha_i + (1 - \tau)\alpha_{i-1}.$$

# Conditioning and loss in precision

- A very important concept is that of a condition number. Badly conditioned system will be difficult or impossible to solve. For linear systems we can make this concrete:
- Define a matrix norm as follows

$$\|A\| = \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|}.$$

- The condition-number of a matrix $A$ is defined by

$$\kappa(A) = \|A\| \cdot \|A^{-1}\|.$$

# Conditioning and loss in precision (cont)

For small perturbations in linear systems, we have that if

$$A(x + \delta_x) = b + \delta_b$$

then

$$\frac{\|\delta_x\|}{\|x\|} \leq \kappa(A) \frac{\|\delta_b\|}{\|b\|}$$

# Conditioning and loss in precision (concl.)

$$\frac{\|\delta_x\|}{\|x\|} \leq \kappa(A)\frac{\|\delta_b\|}{\|b\|}$$

Why?

$$\frac{\|\delta_x\|}{\|x\|} = \frac{\|\delta_x\|}{\|b\|}\frac{\|b\|}{\|x\|} = \frac{\|A^{-1}\delta_b\|}{\|b\|}\frac{\|Ax\|}{\|x\|} \leq \|A^{-1}\| \cdot \|A\|\frac{\|\delta_b\|}{\|b\|}$$

If $b$ has t digits of accuracy then $x$ has $t - \log(\kappa(A))$ digits of accuracy. In this sense $\log(\kappa)$ measures the loss of precision!

# Derivatives

In many numerical methods, we need to evaluate not only a non-linear function but also its first (and perhaps even second) derivatives.
In practice we often use a numerical approximation to the derivative, the so-called finite difference derivative:

$$\frac{\partial f(x)}{\partial x_i} \simeq \frac{f(x + he_i) - f(x)}{h},$$

where $e_i$ is the i'th column of the identity matrix and $h$ is an appropriate step-length.
This is also sometimes refereed to as 'one-sided' finite differences and there might be situations where it is better to use two-sided finite-differences defined as

$$\frac{\partial f(x)}{\partial x_i} \simeq \frac{f(x + he_i) - f(x - he_i)}{2h}.$$

# Automatic Differentiation

- Any high-school student can take derivatives, so we should be able to teach it to a computer.
- There are programs that do automatic forward differentiation (just google "automatic differentiation")
- Tensorflow...

# Linear programming

We consider the following linear minimization problem under linear constraints.

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & c^T x \\ \text{s.t.} \quad & Ax + b \leq 0 \\ & x \geq 0, \end{aligned}$$

where $c \in \mathbb{R}^n$ non-zero, $A$ is an $m \times n$ matrix and $b$ is a $m$-vector. The dual problem to the above is

$$\begin{aligned} \max_{\mu \in \mathbb{R}^m} \quad & b^T \mu \\ \text{s.t.} \quad & A^T \mu + c \leq 0 \\ & \mu \geq 0. \end{aligned}$$

# Linear programming (cont.)

- Simplex method
- Interior point method(s)

# Non-linear systems

- We consider a system of equations

$$f(x) = 0, \quad f : U \subset \mathbb{R}^n \to \mathbb{R}^n$$

- We will usually assume that $f$ is $C^2$ although often/sometimes the methods go through with $f$ being continuous.
- In the economic applications, we will make assumptions that ensure that there is some $\bar{x}$ with $f(\bar{x}) = 0$ and with $D_x f(\bar{x})$ being invertible.
- If $\bar{x}$ satisfies the latter condition, we call the solution **locally unique**.
- Sometimes methods work without this, but if Jacobian is ill-conditioned, we're typically in trouble

# Newton's method

- Suppose that $f : \mathbb{R}^n \to \mathbb{R}^n$ is continuously differentiable.
- Let $z^*$ satisfy $f(z^*) = 0$.
- Suppose $D_z f(z)$ is non-singular, define

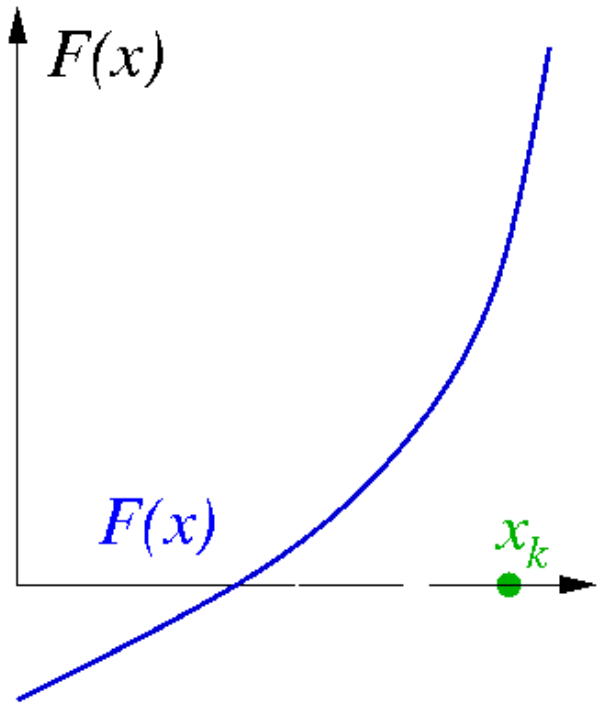
$$N_f(z) = z - (D_z f(z))^{-1} f(z)$$

- Given some $z_0$ define a sequence $(z_i)$ by

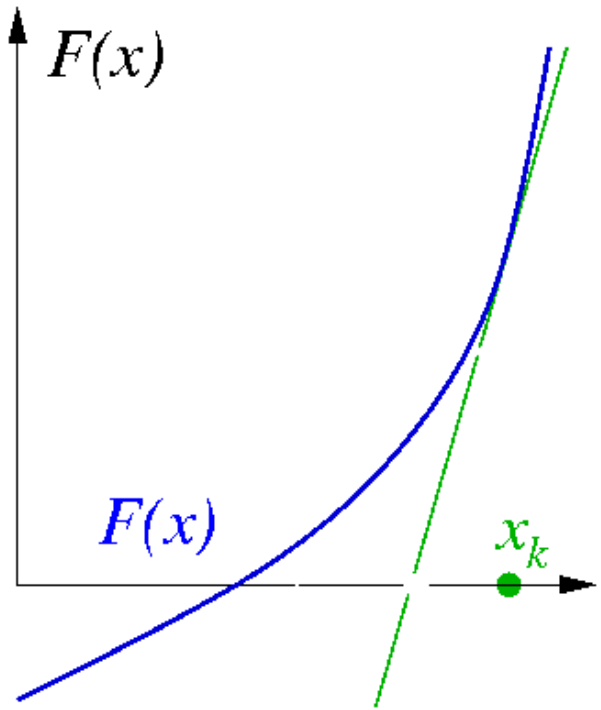
$$z_{i+1} = N_f(z_i), \quad i = 0, ...,$$

  assuming that $D_z f$ is non-singular on the entire sequence.
- If $z_0$ 'sufficiently close to' $z^*$ then $z_i$ converges to $z^*$.

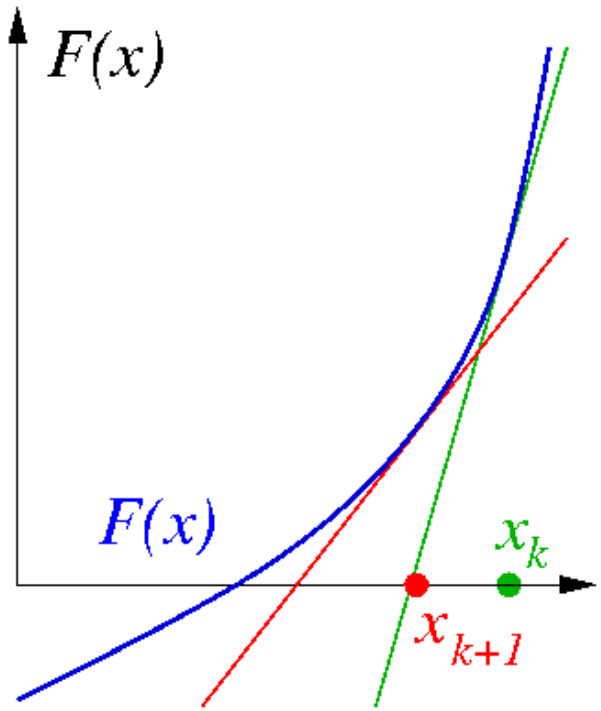
# Newton



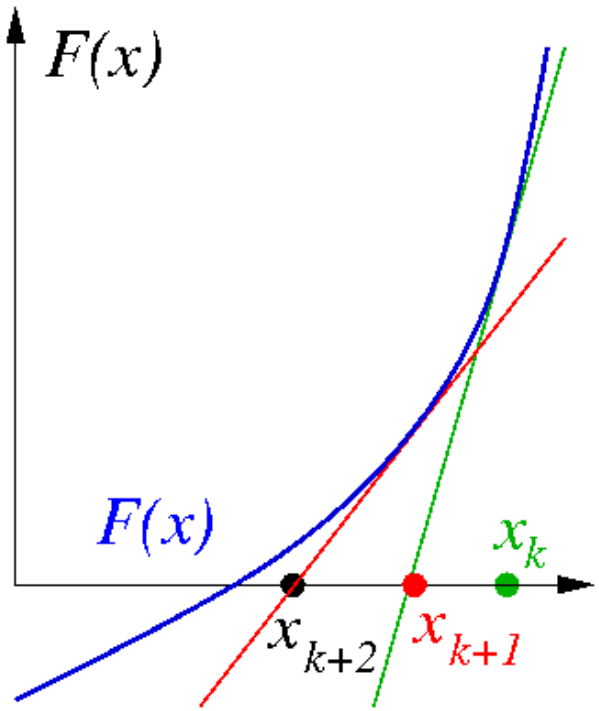
# Newton



# Newton



# Newton



# Better convergence for NM - univariate case

- Note that for the one-variable case if we know that $f(a) \leq 0$ and $f(b) \geq 0$ for some $a < b$ it is trivial to find a zero in $[a, b]$ via bisection: If $f(\frac{a+b}{2}) \leq 0$ let $a = \frac{a+b}{2}$, otherwise let $b = \frac{a+b}{2}$ and repeat. T
- his can be combined with Newton's method by discarding a Newton-iterate whenever it lies outside of the interval in which it is known a root lies. Eventually, Newton's method will make the bisection step unnecessary and we get the quadratic convergence (which is much faster than the linear convergence of bisection).
- Unfortunately this does not generalize to several dimensions

# Better convergence for NM

- Obviously one can view the quest of finding an $x^*$ such that $f(x^*) = 0$ as

$$\min_x \tilde{f}(x) = \frac{1}{2} f(x)^T f(x)$$

- Denote the gradient of $\tilde{f}$ by $g$ i.e.

$$g(x) = D_x \tilde{f}(x) = (D_x f(x))^T f(x).$$

- We write NM somewhat differently:

$$x_{i+1} = x_i + \alpha_i p_i$$

where $\alpha_i$ is a scalar that gives us the step-length and $p_i$ gives the direction. Above we obviously have

$$D_x f(x_i) p_i = -f(x_i)$$

# Better convergence for NM (cont.)

- As in the one-dimensional case, we want to ensure that the Newton-step decreases $\tilde{f}$, i.e. we impose

$$\frac{g(x_i)^T p_i}{\|g(x_i)\| \, \|p_i\|} \leq -\epsilon$$

  for some $\epsilon > 0$.
- We also want to ensure that the step-length $\alpha_i$ is neither too large (and takes us away from the solution) or too small. One way to do this is to impose

$$|g(x_i + \alpha p_i)^T p_i| \leq -\eta g(x_i)^T p_i$$

- It can be shown that (under some additional mild conditions) this method gives a sequence of $x_i$ such that $g(x_i) \to 0$. If $D_f(x_i)$ is invertible this implies that $f(x) = 0$ and we are done.

# Better convergence for NM (concl.)

- It is strong assumption to impose invertibility of $D_x f$ everywhere (it is not enough to have it at $x^*$, obviously, since we might not reach $x^*$).
- The idea out is as follows: iOne can obtain the Newton step, $p_i$ by solving

$$(D_x f(x_i))^T D_x f(x_i) p_i = -(D_x f(x_i))^T f(x_i)$$

  if the Jacobian is non-singular. If it is singular, one can solve instead

$$\left((D_x f(x_i))^T D_x f(x_i) + \lambda_i I\right) p_i = -(D_x f(x_i))^T f(x_i).$$

- For $\lambda_i > 0$ this always has a unique solution and if on picks the $\lambda_i$ cleverly this can lead to $x^*$.
  Homotopy methods...

# NM - Practical issues

- Always important to have a good starting point. It pays to first solve simple problems and use solution as starting point to more complicated ones
- Evaluation of the Jacobian can be very costly. Even if analytic derivatives are available
- Alternative is to use quasi-Newton methods (e.g. Broyden) that do not need the Jacobian at every step

# Constrained optimization

Given $f : U \subset \mathbb{R}^n \to \mathbb{R}$ we consider:

- Unconstrained optimization:

$$\min_{x \in \mathbb{R}^n} f(x)$$

- Box-constrained optimization

$$\min_{x \in \mathbb{R}^n} f(x) \text{ s.t. } l \leq x \leq u$$

- Constrained optimization

$$\min_{x \in \mathbb{R}^n} f(x) \text{ s.t. } c(x) \leq 0$$

# Gradient steepest descent methods

- Fix a step-length, $s$ and just go along the direction where function is decreasing:

$$x_{i+1} = x_i - D_x f(x_i) s$$

- Can do very badly, e.g. Rosenbrock function

$$f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$$

- Not obvious how to pick $s$

# Newton's method for minimization

- Use Newton's method to find $\bar{x}$ such that $D_f(\bar{x}) = 0$
- Denote by $Hf(x)$ the $n \times n$ matrix of second derivatives (Hessian). Then

$$x_{i+1} = x_i - Hf(x_i)^{-1} D_x f(x_i)$$

- Another interpreation: Use Taylor's theorem to approximate $f(.)$ locally by a quadratic function. Take minimum of this function:

$$f(x) \simeq f(x_i) + D_x f(x_i)^T (x - x_i) + \frac{1}{2}(x - x_i)^T Hf(x_i)(x - x_i)$$

$$\min_s f(x_i) + D_x f(x_i)^T s + \frac{1}{2} s^T Hf(x_i) s$$

For convex function we obtain

$$Hf(x_i)s_i = -D_x f(x_i), \quad x_{i+1} = x_i + s_i$$

# Newton's method

- Newton's method translates directly into a method for minimization
- If $f(.)$ is not convex, we can get stuck at stationary points.
  - Trust region methods
  - Line search
- Need to compute the Hessian (can be relaxed and Hessian does not need to be computed every iteration)
- Only find local minimum
- $\rightarrow$ derivative free methods and stochastic optimization

# Trust region method

In each step, $i$, fix a 'trust-region' radius, $\Delta_i$, and minimize the approximating quadratic function under the constraint that $\|s\|_2 \leq \Delta_i$:

$$\begin{aligned} \min_s \quad & f(x_i) + s^T D_x f(x_i) + \tfrac{1}{2} s^T Hf(x_i) s \\ \text{s.t.} \quad & \|s\|_2 \leq \Delta_i \end{aligned}$$

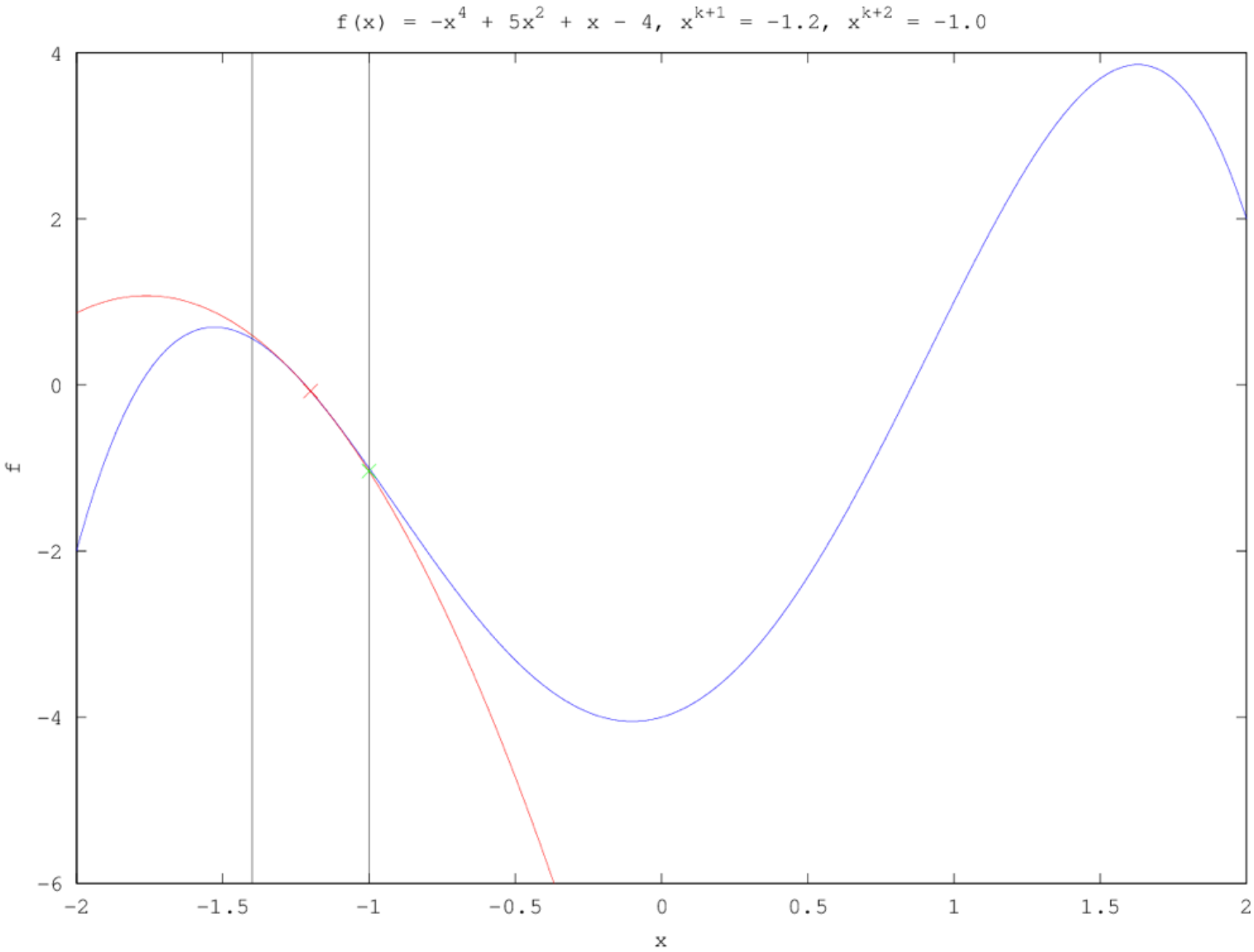
This is a quadratic program that can be solved very efficiently even if $Hf(x_i)$ is not definite. (E.g. conjugate gradient method gives a direction and one can easly analytically minimize the quadratic along this direction)

# Trust region method (cont.)

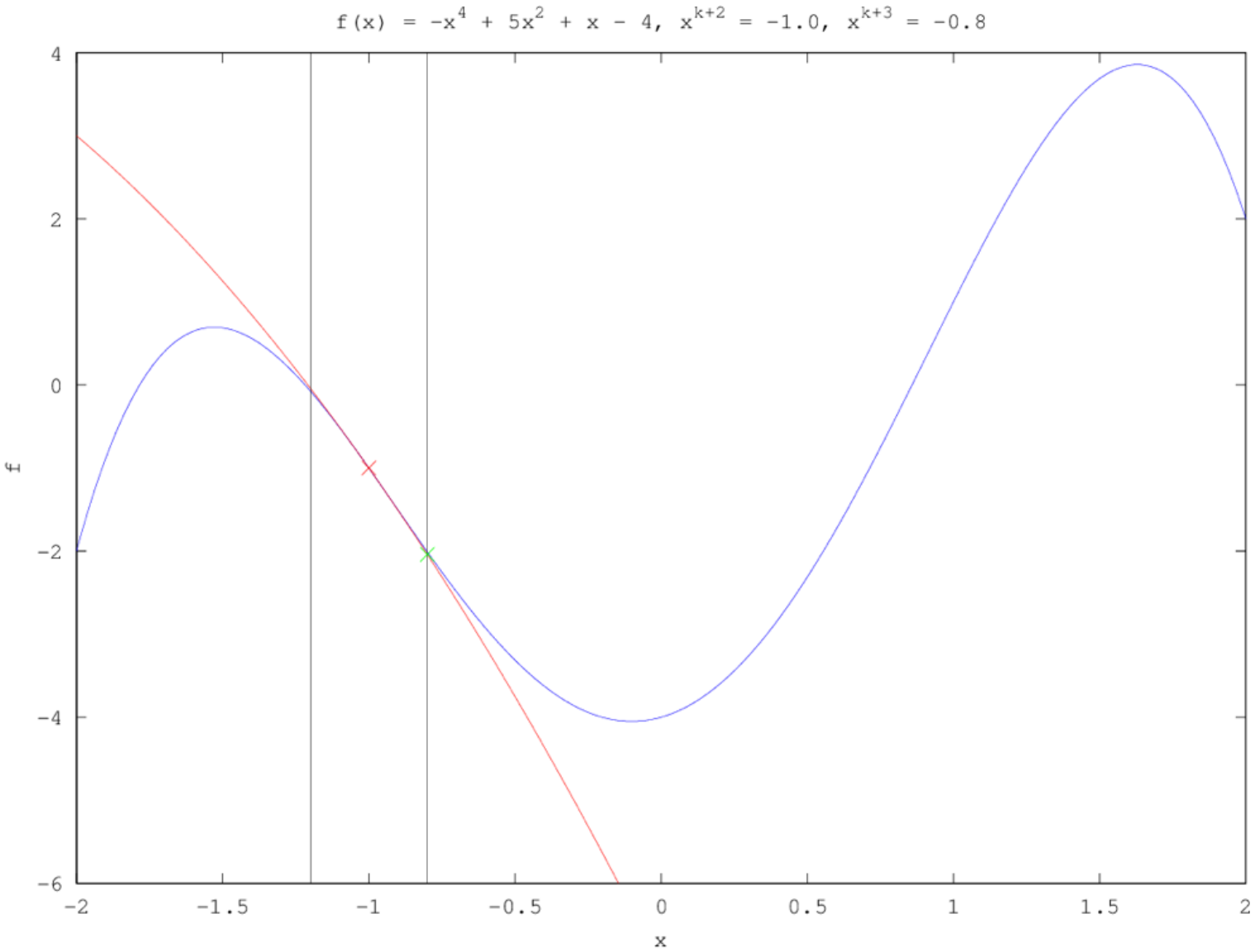
Given initial trust region $\Delta_i$

- Compute new iterate:
  - Solve trust region subproblem
  - Accept or reject the new iterate depending on whether function value decreases or increases
  - Update trust-region radius depending on whether actual reduction more 'than predicted'

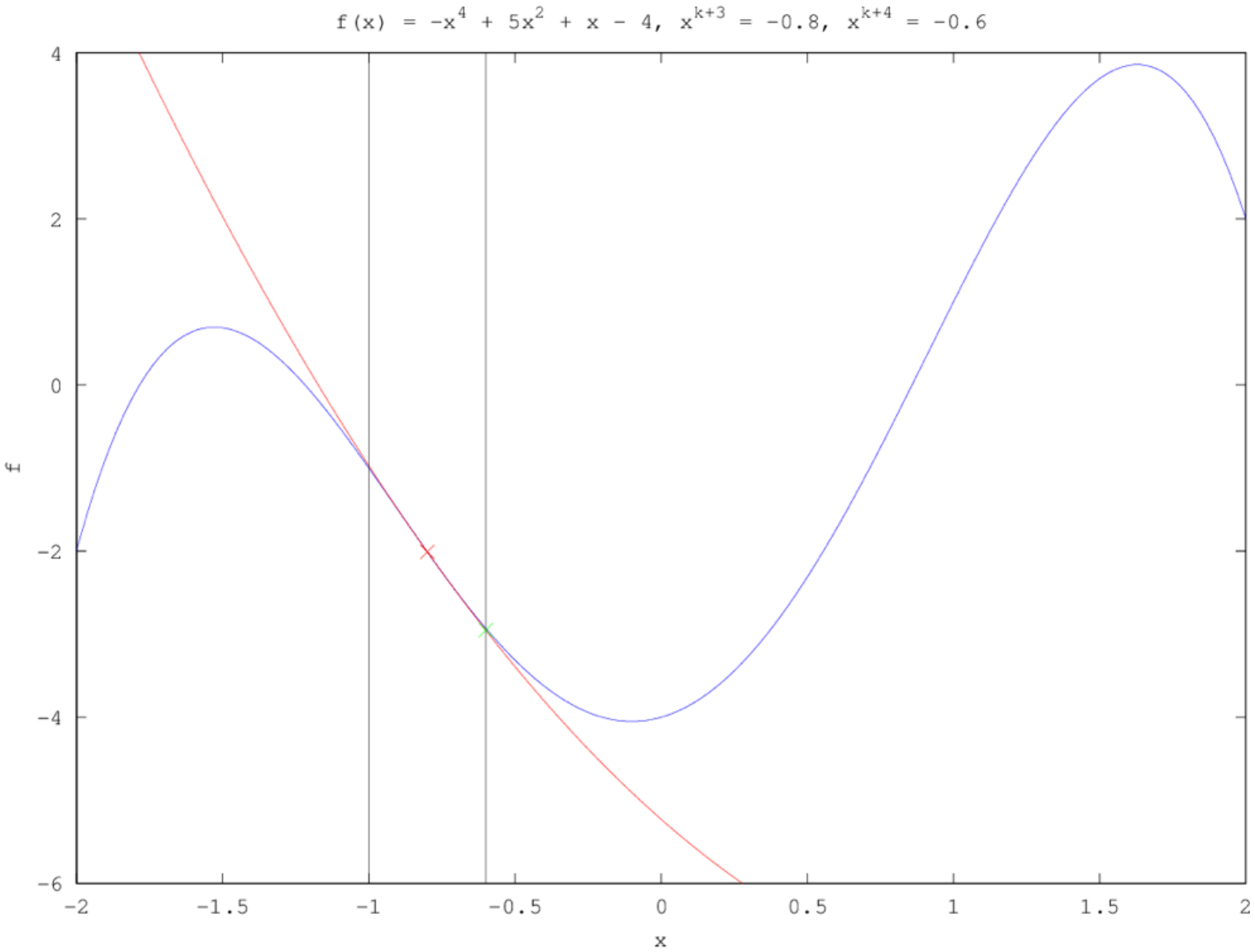
# Example



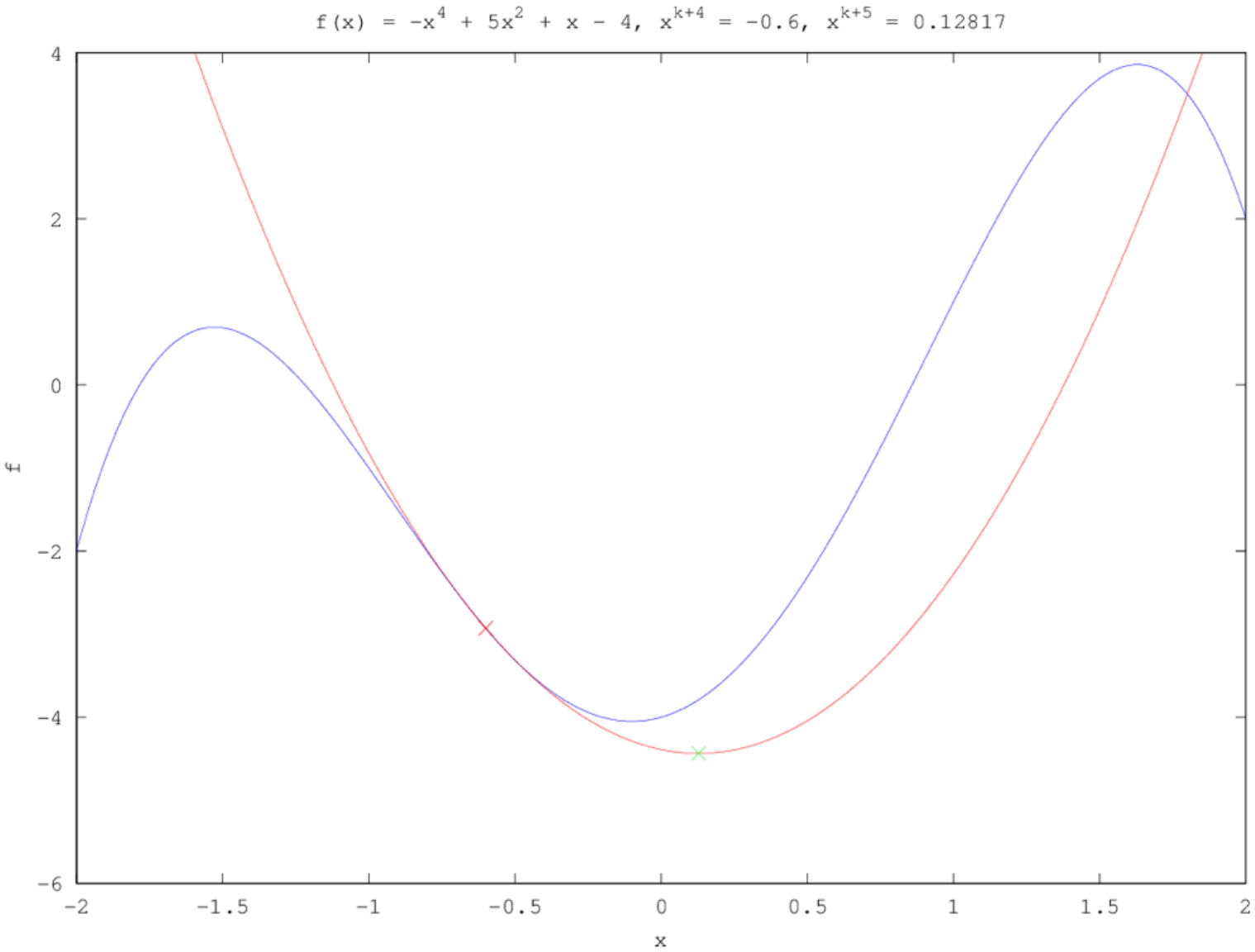
# Example



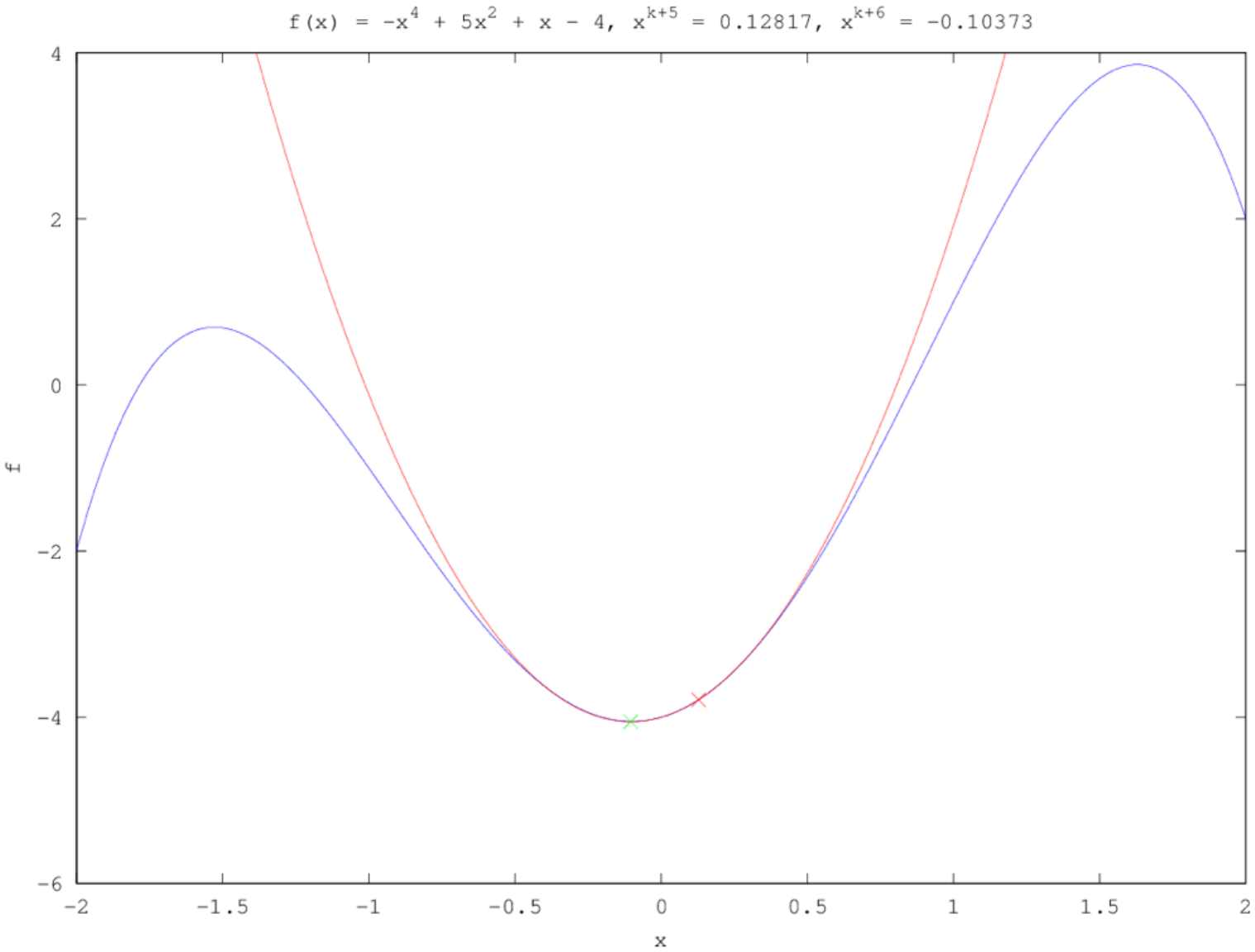
# Example



# Example



# Example



# Constrained optimziation

$$\min_{x \in \mathbb{R}^n} f(x) \text{ s.t. } c(x) \leq 0$$

- As before, good case is $f(.)$ and $\{x : c(x) \leq 0\}$ convex
- In other cases can also look for stationary points. Define Lagrangian
$$\mathcal{L}(x, \lambda) = f(x) + \lambda^T c(x)$$
- Recall Karush-Kuhn-Tucker theorem and the role of constraint qualification

# Constrained optimziation

$$\min_{x \in \mathbb{R}^n} f(x) \text{ s.t. } c(x) \leq 0$$

Many different methods available to solve them. Probably the most common/efficient local methods are

- Sequential (linear-)quadratic programming
- Interior point methods
- Augmented Lagrangian

# Covariance Matrix Adaption Evolutionary strategy

- CMA-ES (Covariance Matrix Adaptation Evolution Strategy) is an evolutionary algorithm for difficult non-linear non-convex black-box optimisation problems in continuous domain.
- A good reference in Nikolaus Hansen (2016, “The CMA Evolution Strategy: A Tutorial”).
- The CMA-ES is can applied to unconstrained or bounded constraint optimization problems, and search space dimensions between three and a hundred.
- If second order derivative based methods are successful, they are faster than the CMA-ES
- The CMA-ES does not use or approximate gradients and does not even presume or require their existence. This makes the method feasible on non-smooth and even non-continuous problems