Intro. Comp. for Data Science (FMI08)

Dr. Nono Saha

June 14, 2023

Max Planck Institute for Mathematics in the Sciences
University of Leipzig/ScaDS.AI

Spring 2023

Course plan

1. Numerical optimization - line search
2. Gradient descent w/ backtracking
3. Homework 6
4. Newton's method
5. Conjugation gradient algorithm
6. More optimization methods: CG

Numerical optimization - line search

Numerical optimization - line search

Today, we will discuss one particular approach for numerical optimization - line search. It is a family of algorithmic approaches that attempt to find (global or local) minima via iteration on an initial guess. Generally, they are an attempt to solve,

$$\min_{\alpha>0} f(x_k + \alpha p_k)$$

where $f(.)$ is the function we are attempting to minimize, x_k is our current guess at iteration k and α is the step length and p_k is the direction of movement.

We will only be dipping our toes in the water of this area, but the goal is to provide some context for some of the more common (and more accessible) use cases. With that in mind, we will look at methods for smooth functions (2nd derivative exists and is continuous).

Line search: gradient method algorithm

Here is an example gradient method that uses a line search in step 4.

1. Set iteration counter $k = 0$, and make an initial guess $\mathbf{x}_0$ for the minimum
2. Repeat:
3. Compute a descent direction $\mathbf{p}_k$
4. Choose α_k to 'loosely' minimize $h(\alpha_k) = f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)$ over $\alpha_k \in \mathbb{R}_+$
5. Update $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$, and $k = k + 1$
6. Until $\|\nabla f(\mathbf{x}_{k+1})\| <$ tolerance

At the line search step (4), the algorithm might either exactly minimize h, by solving $h(\alpha_k) = 0$, or loosely, by asking for a sufficient decrease in h.

Copied from Wikipedia

Exercise: 1d gradient descent algorithm in Python

Given the algorithm previously described, write a function called `grad_desc_1d` that takes four positional arguments:

1. x_0: the initial guess of the minimum
2. f: any continuous function defined from $\mathbb{R} \rightarrowtail \mathbb{R}$ to minimise. It means the argument should be callable.
3. `grad`: a function computing the gradient (or the first derivative) of the function f subjects to minimization.
4. α: the learning rate or the step size

and two optional arguments `max_step` and `threshold` with respective default values: 100 and $1e-6$.

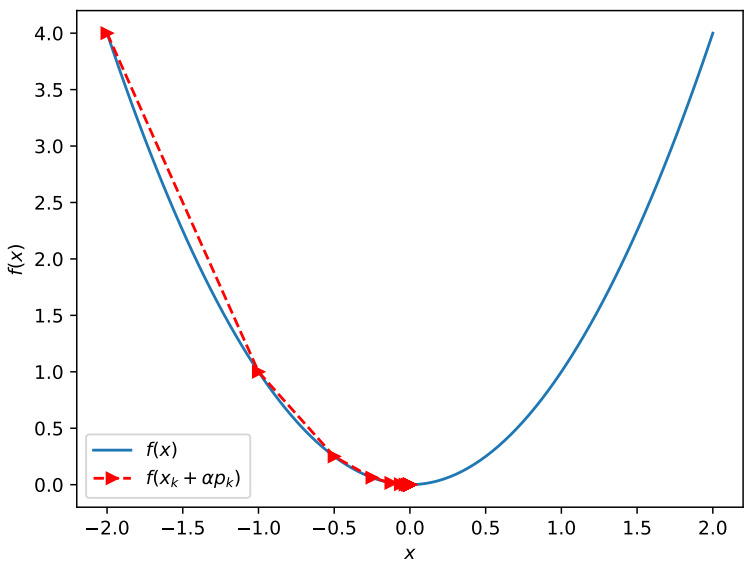
The function should return tow `lists`: x_k and $f(x_k)$.

After writing it, test the function on the two following functions:

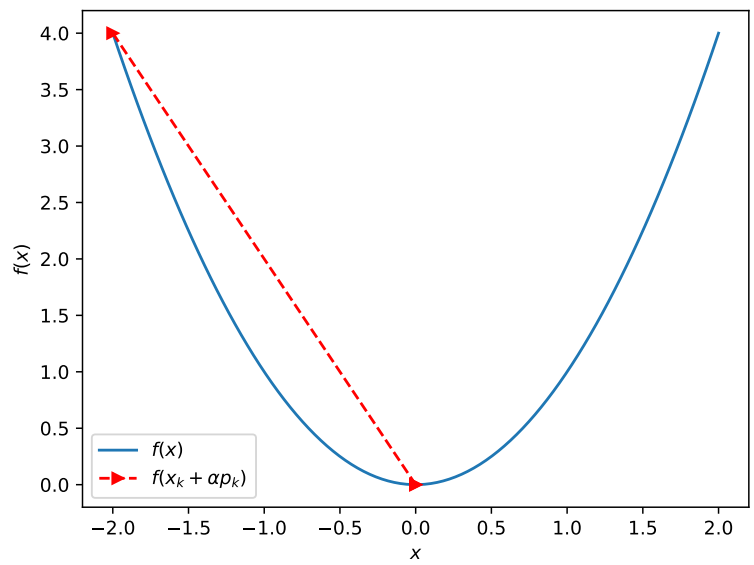
1. $f_1(x) = x^2$
2. $f_2(x) = x^4 + x^3 - x^2 - x$

Gradient descent: our first basic example

```
opt = grad_desc_1d(-2., f, grad,
      alpha=0.25)
plot_1d_traj(-2, 2, f, opt )
```

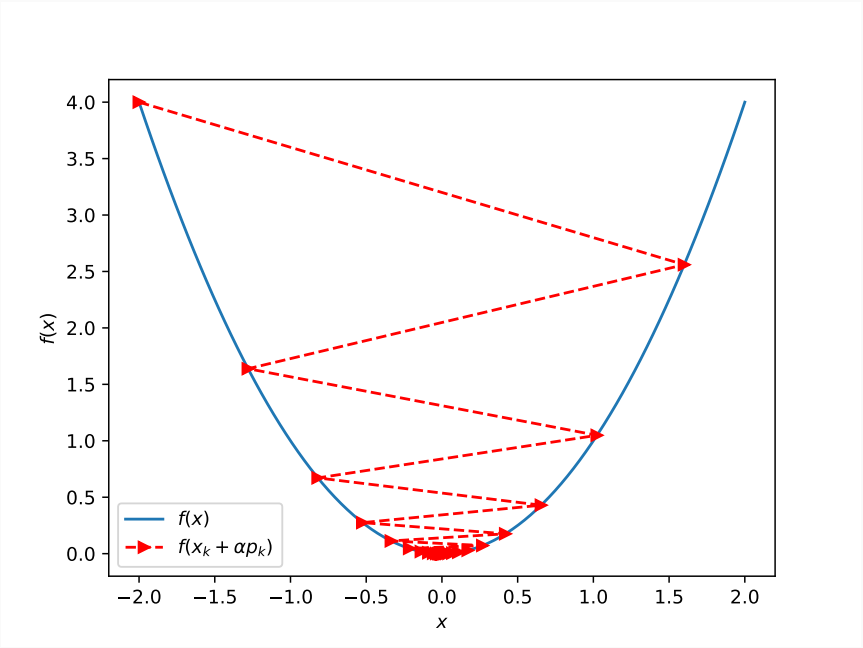



```
opt = grad_desc_1d(-2., f, grad,
      alpha=0.5)
plot_1d_traj(-2, 2, f, opt )
```

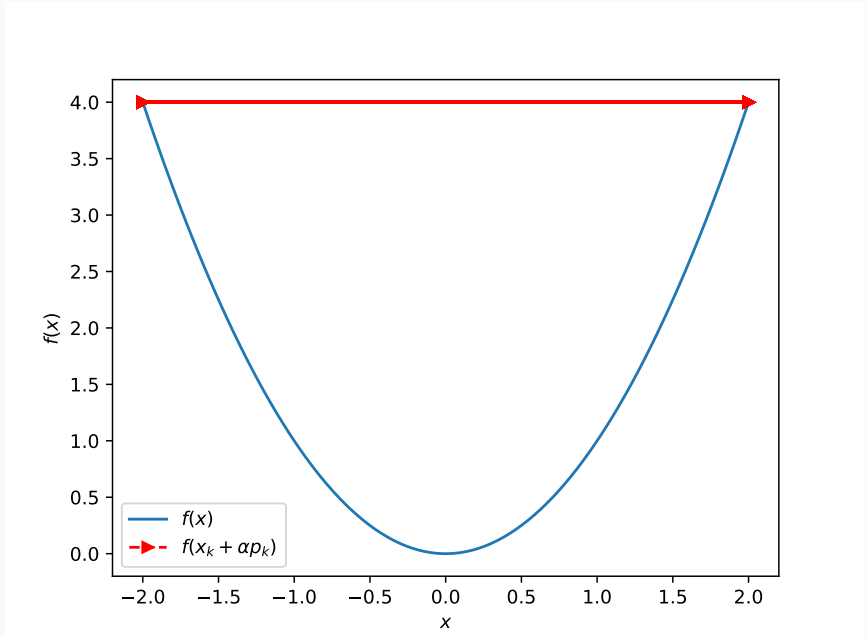


Gradient descent: where can it go wrong?

```
opt = grad_desc_1d(-2., f, grad,
      alpha=0.9)
plot_1d_traj(-2, 2, f, opt )
```




```
opt = grad_desc_1d(-2., f, grad,
      alpha=1)
## Warning - Failed to converge!
plot_1d_traj(-2, 2, f, opt )
```



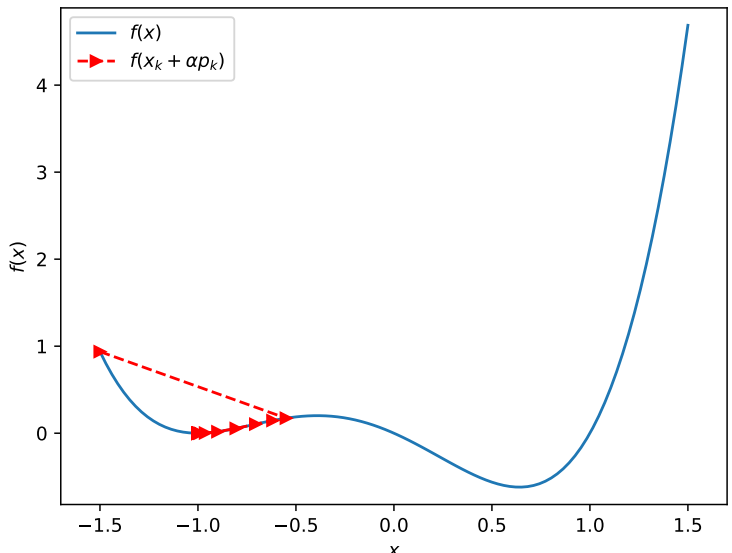
Gradient descent: local minima problem

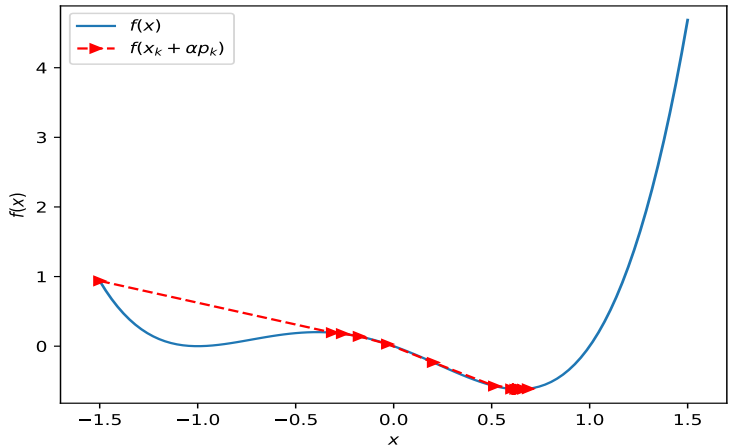
$$f_2(x) = x^4 + x^3 - x^2 - x$$

$$\nabla f_2(x) = 4x^3 + 3x^2 - 2x - 1$$

```
    opt = grad_desc_1d(-2., f2
, grad2, alpha=0.2)
    plot_1d_traj(-1.5, 1.5, f2
, opt)

```

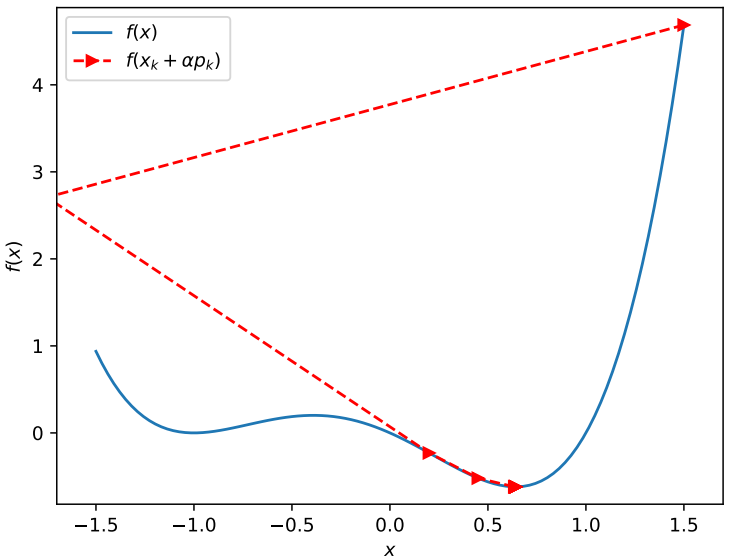
```
f2 = lambda x: x**4 + x**3 - x
    **2 - x
grad2 = lambda x: 4*x**3 + 3*x
    **2 - 2*x - 1

opt = grad_desc_1d(-2., f2,
    grad2, alpha=0.25)
plot_1d_traj(-1.5, 1.5, f2, opt
    )
```



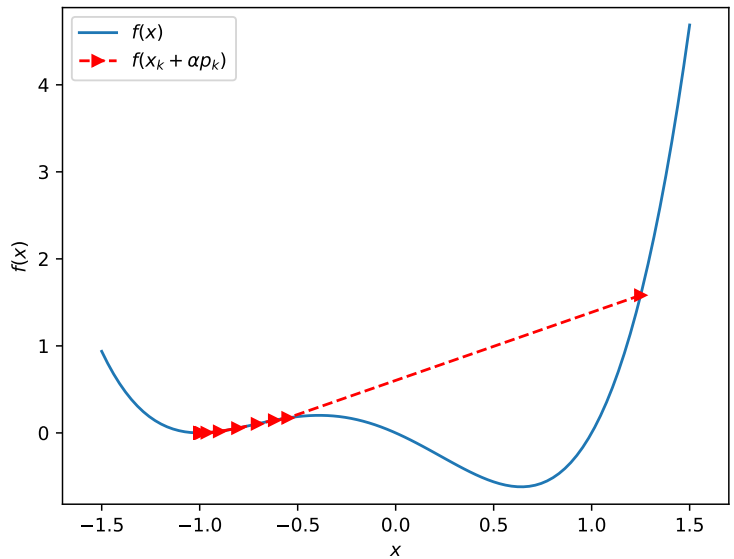


Gradient descent: alternative starting points

```
opt = grad_desc_1d(-1.5, f2,
    grad2, alpha=0.2)
plot_1d_traj(-1.5, 1.5, f2, opt)
```



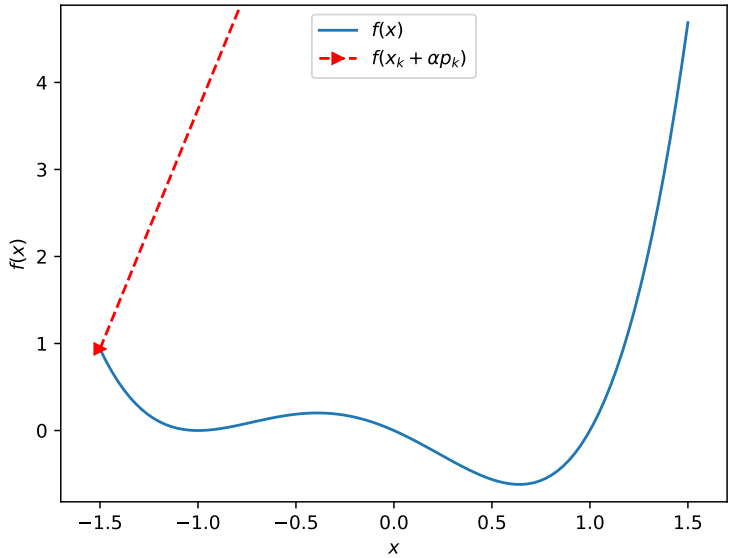

```
opt = grad_desc_1d(-1.25, f2,
    grad2, alpha=0.2)
plot_1d_traj(-1.5, 1.5, f2, opt)
```



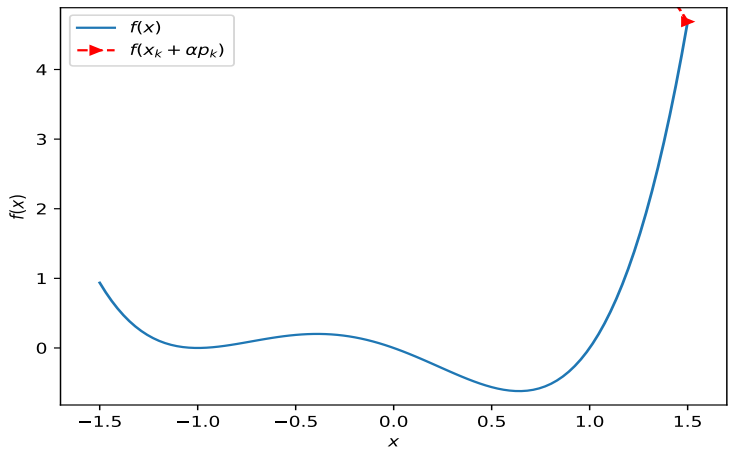
Gradient descent: problematic step sizes

Similarly to the first example, if
the step size is too large, it is impossible for the algorithm to converge.

```
opt = grad_desc_1d(1.5, f2,
    grad2, alpha=0.75)
## OverflowError: (34, 'Result
    too large')
plot_1d_traj(-1.5, 1.5, f2, opt)
```




```
opt = grad_desc_1d(-1.25, f2,
    grad2, alpha=0.2)
## OverflowError: (34, 'Result
    too large')
plot_1d_traj(-1.5, 1.5, f2, opt)
```



Gradient Descent w/ backtracking

Gradient Descent w/ backtracking

As we have just seen that having too large of a step can be problematic, one solution is allowing the size to adapt.

Backtracking involves checking if the proposed move is advantageous (i.e. $f(x_k + \alpha p_k) < f(x_k)$),

- If it is advantageous, then accept

$$x_{k+1} = x_k + \alpha p_k$$

- If not, shrink α by a factor τ (e.g. 0.5) and check again.

Pick larger α to start, as this will not fix the inefficiency of small step size.

This is a hand-wavy version of the Armijo-Goldstein condition. Check $f(x_k - \alpha \nabla f(x_k)) \leq f(x_k) - c\alpha(\nabla f(x_k))^2$.

HomeWork 6: introduction to evolutionary algorithm for CF

In this homework, you will implement your first evolutionary algorithm for solving a continuous function optimization problem. The variant of the evolutionary algorithm you will implement is called the differential evolutionary algorithm, and the algorithm follows the steps:

1. Generate the initial population of N individuals or agents P_t.
2. Evaluate the Fitness of each individual
3. Mutate the population P_t to generate a new population P'_t
4. Select N individual from $P_t \cup P'_t$ proportianally to their fitnesses. The selected individuals will reproduce to the next generation and form a new population P_{t+1}
5. Stop if the halting criterion is satisfied; otherwise, $t = t + 1$ and go to step 2.

HomeWork 6: EA initialisation algorithm

For the simple and classical implementation of DEA, the population consists of $N-$individuals taken as a pair of real-valued vectors, $(x_i, \eta_i), \forall i \in \{1, 2, \ldots, N\}$. x_i's are objective variables, and η_i's are standard deviations for Gaussian mutations (also known as strategy parameters in self-adaptive evolutionary algorithms).

For a given interval x^L and x^U, the initial population is

$$P_t = \{X_i\}, \forall i \in \{1, 2, \ldots, N\}$$

where $X_i = (x_i, \eta_i)$ and

- $x_i = r_i(x^U - x^L) + x^L$, where $r_i \sim \mathcal{U}(0, 1)$
- $\eta_i = r'_i(x^U - x^L) + x^L$, where $r'_i \sim \mathcal{U}(0, 1)$

Note that x_i is an n-dimensional vector. i.e. $x_i \in \mathbb{R}^n$.

HomeWork 6: EA evaluation step

In this homework, you will test your implementation on three functions:

1. $f_1(x) = x^4 + x^3 - x^2 - x$, for $n = 1$ and $x^L = -2, x^U = 2$
2. $f_2(x) = \sum_{j=1}^{n} x_j$, for $n = 30$ and $x^L = -100, x^U = 100$
3. $f_3(x) = \sum_{j=1}^{n} [100(x_{j+1} - x_j^2)^2 + (x_j - 1)^2]$, for $n = 30$ and $x^L = -30, x^U = 30$.

Note that in all three cases, the minimum value of the function is 0, which means the evaluation of the best individual should be 0 in a successful optimization case.

One of the expensive steps in a DEA is to evaluate the individual's fitness in the population. This allows us to compute the reproduction rate or implement the so-called "natural selection". The evaluation consists of simply computing the value of one of the above functions (or objective function) at the given individual point x_i.

HomeWork 6: EA mutation step

Mutation operation allows diversity in the population or, in our optimization jargon, to generate new solutions (or new x_i). So, for each parent $(xi, \eta_i), i = 1, \ldots, N$, creates a single offspring (x_i', η_i') by:

For $j = 1, \ldots, n$

$$x_i'(j) = x_i(j) + \eta_i(j) \times \mathcal{N}_j(0, 1) \tag{1}$$

$$\eta_i'(j) = \eta_i(j) \exp\left(\tau' \times \mathcal{N}(0, 1) + \tau \times \mathcal{N}_j(0, 1)\right) \tag{2}$$

where $x_i(j), x_i'(j), \eta_i(j)$, and $\eta_i'(j)$ denote the $j - th$ component of the vectors $x_i(j), x_i'(j), \eta_i(j), \eta_i'(j)$, respectively. The factors τ and τ' are commonly set to $(\sqrt{2\sqrt{n}})^{-1}$ and $(\sqrt{2n})^{-1}$.

HomeWork 6: EA selection step

The operator consists of selecting solutions in the population for reproduction. The fitter the solution, the more times as likely it is selected to reproduce. This operator often requires a fitness function evaluation, which means we must evaluate the fitness of the mutated population P'_t.

You will implement here the most straightforward selection operator using the two following steps:

1. Conduct a pairwise comparison over the union of parents (x_i, η_i) and the offspring $(x'_i, \eta'_i), \forall i \in \{1, 2, \dots N\}$. For each individual i, q opponents are chosen uniformly at random from all the parents and offspring. For each comparison, if the individual's fitness is smaller than the opponent's, it receives a "win."
2. Select N individuals out of P_t and P'_t that have the most wins to be parents of the next generation.

Newton’s Method in 1d

Newton's Method in 1d

Lets simplify things for now and consider just the 1d case and write αp_k as ∇,

$$f(x_k + \Delta) \approx f(x_k) + \Delta f'(x_k) + \frac{1}{2}\Delta^2 f''(x_k)$$

To find the Δ that minimizes this function, we can take a derivative with respect to Δ and set the equation equal to zero, which gives,

$$0 = f'(x_k) + \Delta f''(x_k) \Rightarrow \Delta = \frac{f'(x_k)}{f''(x_k)}$$

Which then suggests an iterative update rule of

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

Newton's Method: generalizing to nd

Based on the same argument, we can see the following result for a function in $\mathbb{R}^n$,

$$f(x_k + \Delta) \approx f(x_k) + \Delta^T \nabla f(x_k) + \frac{1}{2}\Delta^T \nabla^2 f(x_k)$$

To find the ∇ that minimizes this function, we can take a derivative with respect to ∇ and set the equation equal to zero, which gives,

$$0 = \nabla f(x_k) + \nabla^2 f(x_k) \Rightarrow \Delta = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k) f(x_k)$$

Which then suggests an iterative update rule of

$$x_{k+1} = x_k - (\nabla^2 f(x_k))^{-1} \nabla f(x_k) f(x_k)$$