INF3490/INF4490 - Week 1 solutions
Simple search algorithms

August 28, 2018

ℙ marks the programming exercises, we strongly recommend using the python programming language for these. Exercises may be added/changed after publishing.

1 Simple search algorithms

Given the function $f(x) = -x^4 + 2x^3 + 2x^2 - x$:

1.a Derivative

What is its derivative $f'(x)$?

Answer:

$$f'(x) = -4x^3 + 6x^2 + 4x - 1$$

ℙ 1.b Plotting

Plot the function, and its gradient(derivative) from $x = -2$ to $x = 3$. Use python, wolfram alpha or another plotting tool of your choice.

Plot:

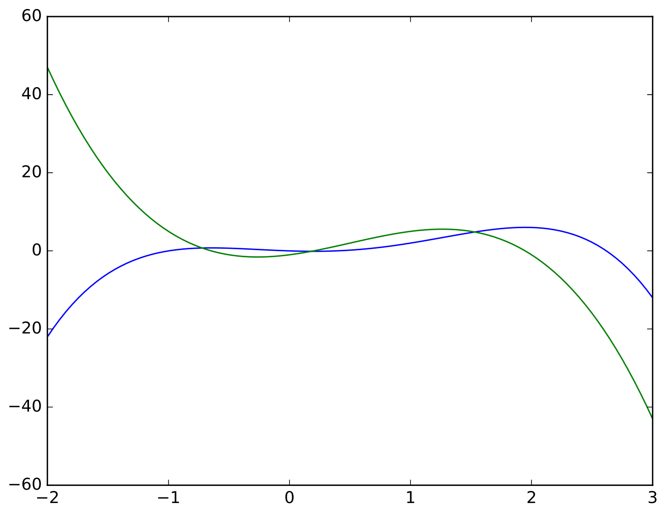

Figure 1: $f(x)$ and it's derivative.

Source code (Python 3):

```
#!/usr/bin/env python3
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return -x**4 + 2 * x**3 + 2 * x**2 - x

def df(x):
    return -4 * x**3 + 6 * x**2 + 4 * x - 1

x = np.linspace(-2, 3, 100)
plt.plot(x,f(x))
plt.plot(x,df(x))
plt.savefig("w1e1b.eps", format="eps")
plt.show()
```

ℙ 1.c Gradient Ascent

Maximize using gradient ascent. You can try step size 0.1 and start somewhere in the range [-2, 3]. How does the choice of starting point and step size affect the algorithm's performance? Is there a starting point where the algorithm would not even be able to find a local maximum?

Answer:
Both starting position and step size affects where the algorithm ends:

- Starting Position
  - *Left side:* Should converge on left maximum
  - *Center:* Stops immediately, gradient is zero.
  - *Right side:* Should converge on right maximum
- Step Size
  - *Too low:* Converges slowly (poor performance)
  - *Too high:* Overshoot, bounce over solutions. Doesn't converge, might not terminate.

Plot:

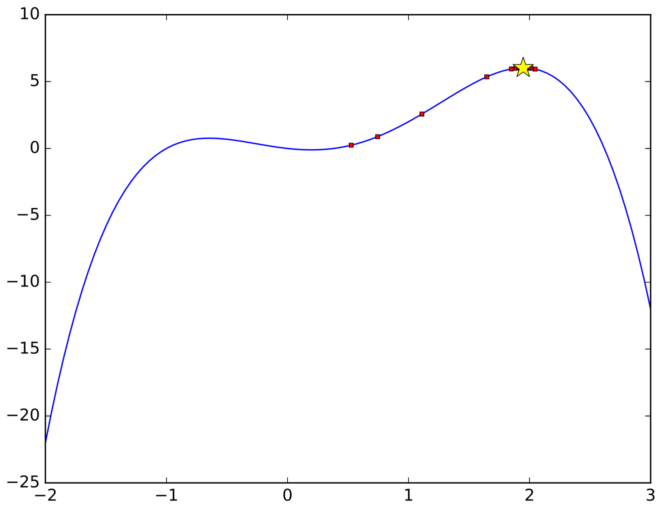

Figure 2: Result of gradient ascent

Source code (Python 3):

```
#!/usr/bin/env python3
import numpy as np
import matplotlib.pyplot as plt
import random

def f(x):
    return -x**4 + 2 * x**3 + 2 * x**2 - x

def df(x):
    return -4 * x**3 + 6 * x**2 + 4 * x - 1

def gradient_ascent(gamma, x, precision):
    dx = gamma * df(x)
    while abs(dx) > precision:
        plt.plot(x,f(x), color="red", marker="s", markersize=3)
        x  = x + dx
        dx = gamma * df(x)
    return x,f(x)

def plot_gradient_ascent(start,stop,steps):
    x = np.linspace(start, stop, steps)

    plt.plot(x,f(x))
    randx = random.uniform(start,stop)
    sol = gradient_ascent(gamma=0.1, x=randx, precision=0.0001)
    plt.plot(sol[0],sol[1], color="yellow", marker="*", markersize=16)
    plt.savefig("eps/w1e1c.eps", format="eps")
    plt.show()

if __name__ == "__main__":
    plot_gradient_ascent(-2,3,100)
```

ℙ 1.d Exhaustive Search

Assume that we are only interested in maxima of $f(x)$ where $-2 \leq x \leq 3$, and x increases in steps of length 0.5. ($\Delta x = 0.5$). Perform an exhaustive search to maximize $f(x)$ and plot the result.

Plot:

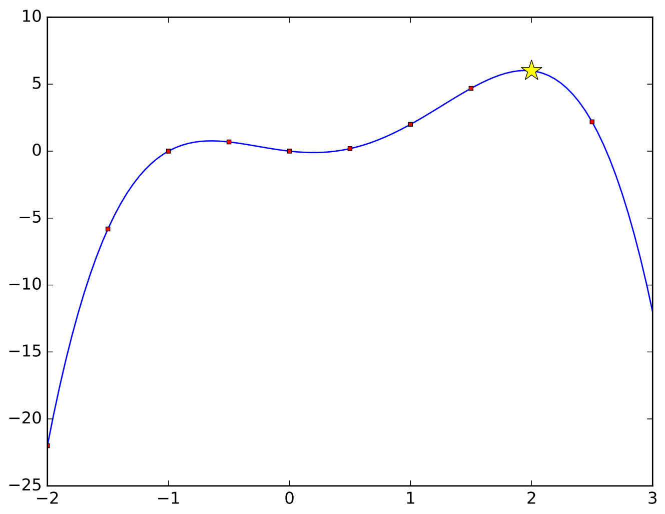

Figure 3: Result of exhaustive search

Source code (Python 3):

```
#!/usr/bin/env python3
import numpy as np
import matplotlib.pyplot as plt
import random

def f(x):
    return -x**4 + 2 * x**3 + 2 * x**2 - x

def exhaustive(function, start, stop, step):
    x = start
    best = (x, function(x))
    while x < stop:
        y = function(x)
        if y > best[1]:
            best = (x,y)
        plt.plot(x,y, color="red", marker="s", markersize=3)
        x += step
    return best

def plot_exhaustive(function,start,stop,steps):
    x = np.linspace(start, stop, steps)

    plt.plot(x,function(x))
    randx = random.uniform(start,stop)
    sol = exhaustive(function, start, stop, step=0.5)
    plt.plot(sol[0],sol[1], color="yellow", marker="*", markersize=16)
    plt.savefig("eps/w1e1d.eps", format="eps")
    plt.show()

if __name__ == "__main__":
    plot_exhaustive(f,-2,3,100)
```

1.e Possible improvements

Gradient ascent and hill climbing are quite similar, and they are based almost exclusively on exploitation. Can you think of any additions to these algorithms in order to do more exploration?

Answer:
Run the algorithm several times with random starting positions, this will *explore* the solution space and find several local optima. Another option is to add more random movement to either algorithm. This can be done after a solution is found, or at a probability while searching. Could also do backtrack + random jump after a solution is found.

1.f Exhaustive search vs. simulated annealing

Which algorithm do you think is the most efficient at maximizing $f(x)$ under the conditions in Problem 1.d: exhaustive search or simulated annealing? Explain.

Answer:
Exhaustive search is better for this case. It takes few iterations to go through the one dimensional solution space. For problems in higher dimensions or more complex functions with smaller step size exhaustive search becomes impractical and simulated annealing will outperform.

Contact

The exercises are the same as those in the **GitHub repository**. If there are any suggestions to corrections of grammar, language or any additional suggestions, we appreciate all feedback! The TA's can be reached at **lonnekes@ifi.uio.no**, **hermankn@ifi.uio.no**, **a.s.skage@econ.uio.no** or **sharanak@ifi.uio.no**