ESI 6341 - Introduction to Stochastic Optimization
Homework 6 - Kelley's Algorithm

James Diffenderfer

April 19, 2018

Contents

Part (a)

I chose to solve the problem

$$\begin{aligned} \min \quad & x \\ \text{s.t.} \quad & x \in [-1, 1]. \end{aligned}$$

Using the initial point $x^0 = 1$ and a stopping tolerance of $1e^{-6}$, our implementation of Kelley's Algorithm reached the stopping criteria in one iteration and returned the solution $x^1 = -1$ which is the global minimum of the objective function with respect to the constraint. Below is a table of values and a graph of the iterates and the objective function.

k	x^k	$f_k(x^k)$	$f(x^k)$
0	1	1	1
1	-1	-1	-1

Figure 1: Table of values determined while performing Kelley's Algorithm

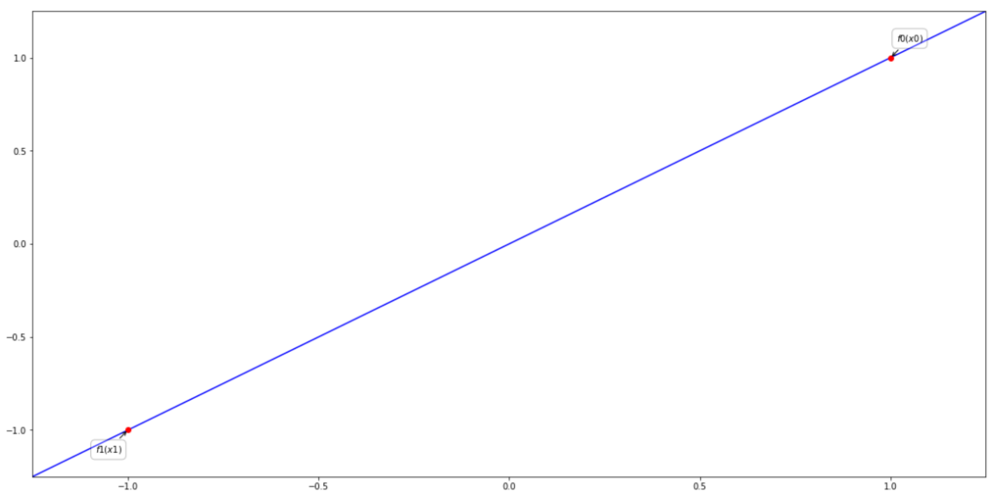

Figure 2: Plot of $f(x) = x$ and $f_k(x^k)$ for $0 \leq k \leq 1$

Part (b)

For this part we were supposed to solve the problem

$$\min \quad (x-2)^2+1 \quad \text{s.t.} \quad x\in[-1,4]$$

with initial point $x^0 = -1$. Using our implementation of Kelley's Algorithm with a stopping tolerance of $1e^{-6}$, the stopping criteria were reached in 14 iterations and returned the solution $x^{14} = 1.99987793$. The exact solution is $x^* = 2$. Below is a table of values and a graph of the iterates and the objective function. Since the last several iterates are fairly close to each other we provided two different plots. The first contains all of the iterates and the second provides a zoomed in view of the last several iterates.

k	x^k	$f_k(x^k)$	$f(x^k)$
0	-1	10.0	10.0
1	4	-20	5.0
2	1.5	-5	1.25
3	2.75	0	1.5625
4	2.125	0.6249999999999998	1.015625
5	1.8125	0.9374999999999998	1.03515625
6	1.96875	0.9765624999999996	1.0009765625
7	2.046875	0.99609375	1.002197265625
8	2.0078125	0.9985351562500001	1.00006103515625
9	1.98828125	0.9997558593750003	1.0001373291015627
10	1.99804688	0.999908447265625	1.0000038146972656
11	2.00292969	0.9999847412109376	1.0000085830688477
12	2.00048828	0.9999942779541019	1.000000238418579
13	1.99926758	0.9999990463256833	1.000000536441803
14	1.99987793	0.9999996423721312	1.0000000149011612

Figure 3: Table of values determined while performing Kelley's Algorithm

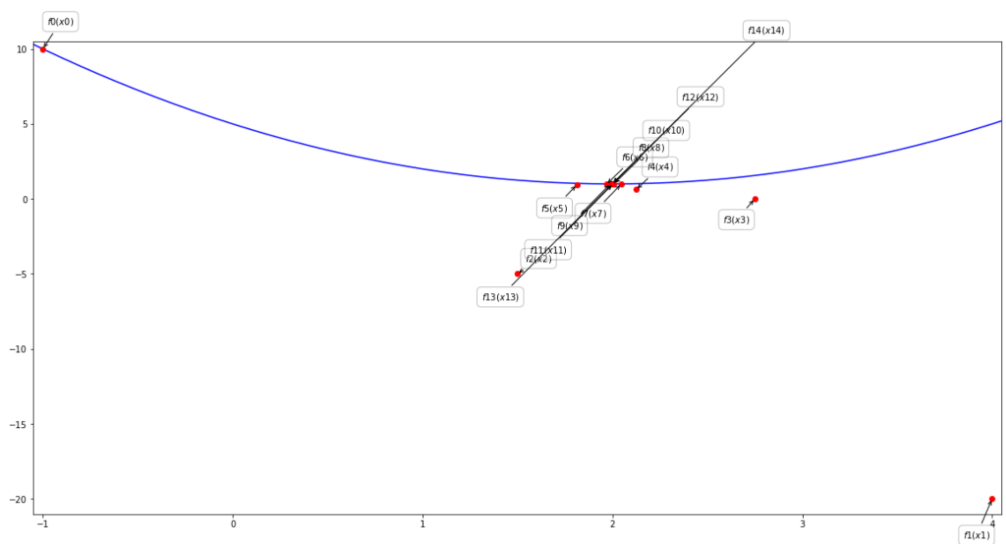

Figure 4: Plot of $f(x) = (x-2)^2 + 1$ and $f_k(x^k)$ for $0 \le k \le 14$

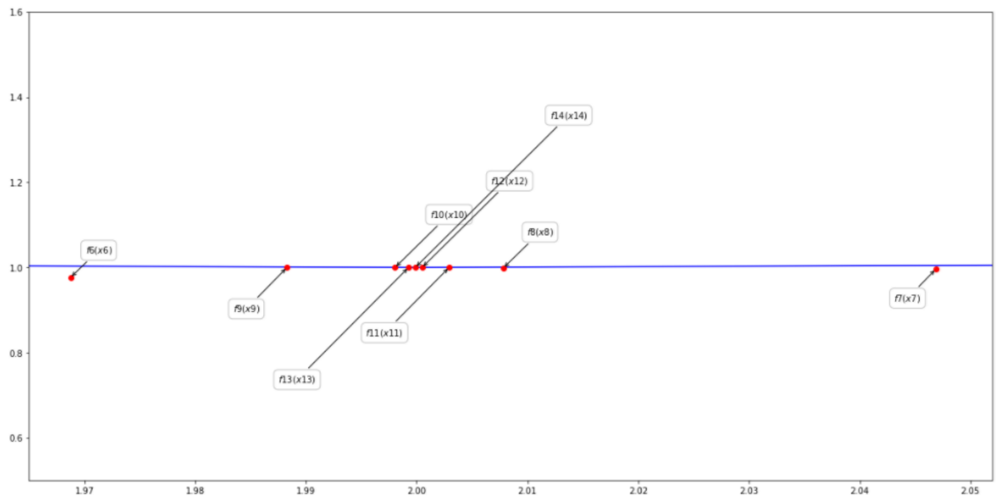

Figure 5: Plot of $f(x) = (x-2)^2 + 1$ and $f_k(x^k)$ for $6 \le k \le 14$

Appendix

Here we provide our code for Kelley's Algorithm. The algorithm was coded using python. We made use of the libraries numpy, scipy, and matplotlib to solve the linear program in Kelley's Algorithm and generate a plot of the points computed at each iteration of the algorithm.

Kelley's Algorithm

```
import numpy as np
from scipy.optimize import linprog
import matplotlib.pyplot as plt

def f(x):
    return x

def fprime(x):
    return 1

def g(x):
    return (x-2)**2 + 1

def gprime(x):
    return 2*x - 4

def kelley(x0, xmin, xmax, f, fprime, eps, max_iters):
    # Initialize array to hold alpha, beta, and x
    alpha = []
    beta = []
    x = []
    lk = []
    # Append initial x value to list
    x = np.append(x, x0)
    # Initialize uk
    uk = f(x[0])
    # Initialize value of c for linprog
    c = np.array([0,1])
    # Initialize bounds for linprog
    x0_bounds = (xmin, xmax)
```

```
    x1_bounds = (None, None)
    # Generate supporting hyperplanes to find minimum
    for k in range(max_iters):
        # Compute beta value
        beta = np.append(beta, fprime(x[k]))
        # Compute alpha value
        alpha = np.append(alpha, f(x[k]) - beta[k] * x[k])
        # ----- Compute eta value -----
        # Initialize A
        A = np.stack((beta, -np.ones(k+1)), axis = 1)
        # Initialize b
        b = -np.copy(alpha)
        # Solve LP to get eta
        eta = linprog(c, A_ub=A, b_ub=b, bounds=(x0_bounds, x1_bounds))
        # Use solution from linear program to store x_{k+1} and lk
        x = np.append(x, eta.x[0])
        lk.append(eta.fun)
        # Update uk value
        uk = min(uk, f(x[k+1]))
        # Update temp
        t = x[k+1]
        # Check stopping criteria
        if (uk - lk[k] <= eps):
            # Stopping criterion satisfied
            print("Stopping criterion satisfied in %i iterations." %(k+1))
            print("Terminating Program and returning minimizer.")
            break
        if k == max_iters - 1:
            print("Maximum number of iterations reached without satisfying stopping criteria.")
            print("Terminating program and returning current value for x_%i." %(k+1))
    # Return solution
    return x, lk

# Run kelley program on problem of my choice
x, lk = kelley(1, -1, 1, f, fprime, 1e-6, 50)
print("x = ", x)
print("lk = ", lk)

# Run kelley program for part (b)
x, lk = kelley(-1, -1, 4, g, gprime, 1e-6, 50)
print("x = ", x)
print("lk = ", lk)

# ---------- Generate Plot -------------
# Set x values
xplt = x[0:len(x)]
lk = np.insert(lk, 0, g(x[0]))

# Set plot size
plt.figure(figsize=(20,10))
# Plot function
t = np.arange(0., 5., 0.1)
plt.plot(t, (t-2)**2 + 1, 'b-')
# Plot iterates and fk values
plt.plot(xplt, lk, 'ro')
# Generate labels for points
labels = [r'$f{0}(x{0})$'.format(i) for i in range(1,len(lk)+1)]
# Place labels on plot
i = 0
for label, a, b in zip(labels, xplt, lk):
    i = i + 1
    plt.annotate(
            label,
            xy=(a, b), xytext=(a - (-1)**i * (20 + 1.4**i), b - (-1)**i * ( 20 + 1.4**i)),
            textcoords='offset points', ha='center', va='center',
```

```
            bbox=dict(boxstyle='round,pad=0.5', fc='white', alpha=0.25),
            arrowprops=dict(arrowstyle = '->', connectionstyle='arc3,rad=0'))
# Set axes for plot
plt.axis([1.2, 2.8, -0.25, 1.8])
# Save figure to file
plt.savefig('kelley_plot.png')
# Show plot
plt.show()
```