Slice Sampling Multimodal

March 4, 2019

1 Multimodal Slice Sampling Example

1.0.1 Import Libraries

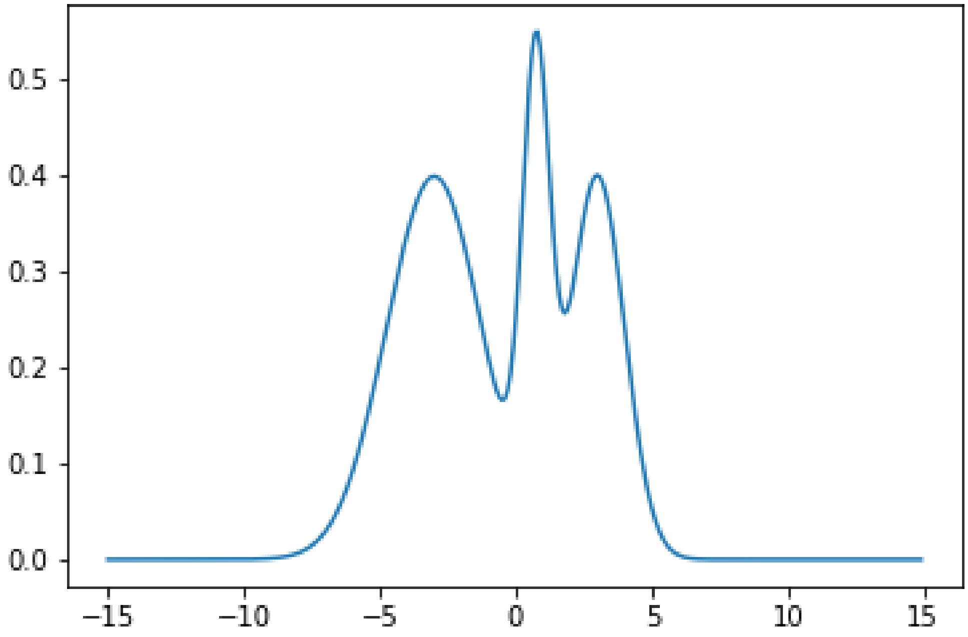
```
In [266]: import numpy as np
          import scipy.stats as stats
          import matplotlib.pyplot as plt
          %matplotlib inline
```

1.0.2 Generate Unimodal Gaussian

```
In [267]: # Parameters used for Gaussian
          x_low = -15
          x_high = 15
          x_step = 0.1

          # Generate pdf
          X = np.arange(x_low, x_high, x_step)
          Y = 1.75*stats.norm.pdf(X,-3,1.75) + .6*stats.norm.pdf(X,.75,.5) + stats.norm.pdf(X,3

          plt.plot(X, Y)
          plt.show()
```



1.0.3 Define function for initial sample for x_0

```
In [270]: low = X[0]
          high = X[-1]

          def __sample_x(low=low, high=high):
              x = np.random.uniform(low=low, high=high)
              return x

          current_x = __sample_x(low, high)

          print(current_x)
```

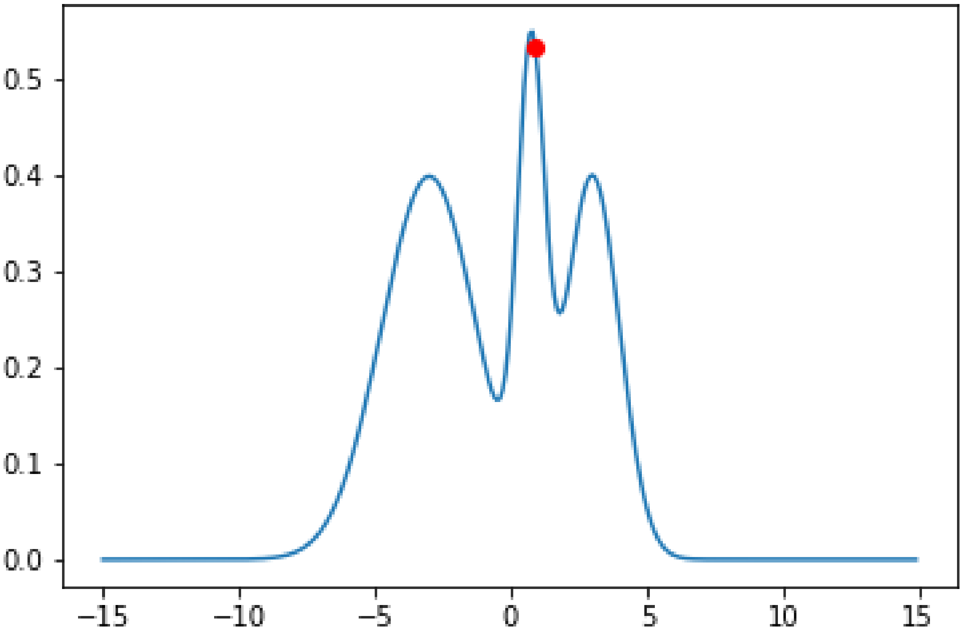
```
0.8666991416777119
```

1.0.4 Define function for calculating f(x_0)

```
In [271]: def __fx(x):
              ind = np.argmin(np.abs(X - x))
              y = Y[ind]
              return y

          current_fx = __fx(current_x)

          plt.plot(X, Y)
          plt.plot([current_x], [current_fx], 'ro')
          plt.show()
```

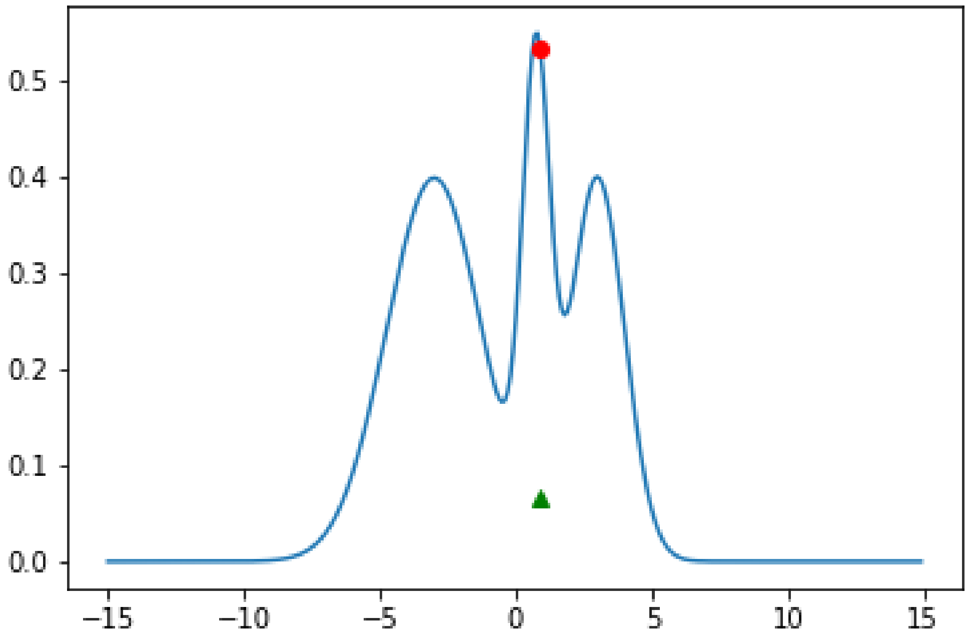


1.0.5 Define function for sampling in the interval (0, f(x_0)), calculate y

```
In [272]: def __sample_y(x, y):
              sampled_y = np.random.uniform(low=0, high=y)
              return sampled_y

          current_sampled_y = __sample_y(current_x, current_fx)


          plt.plot(X, Y)
          plt.plot([current_x], [current_fx], 'ro')
          plt.plot([current_x], [current_sampled_y], 'g^')
          plt.show()
          print(current_sampled_y)
```



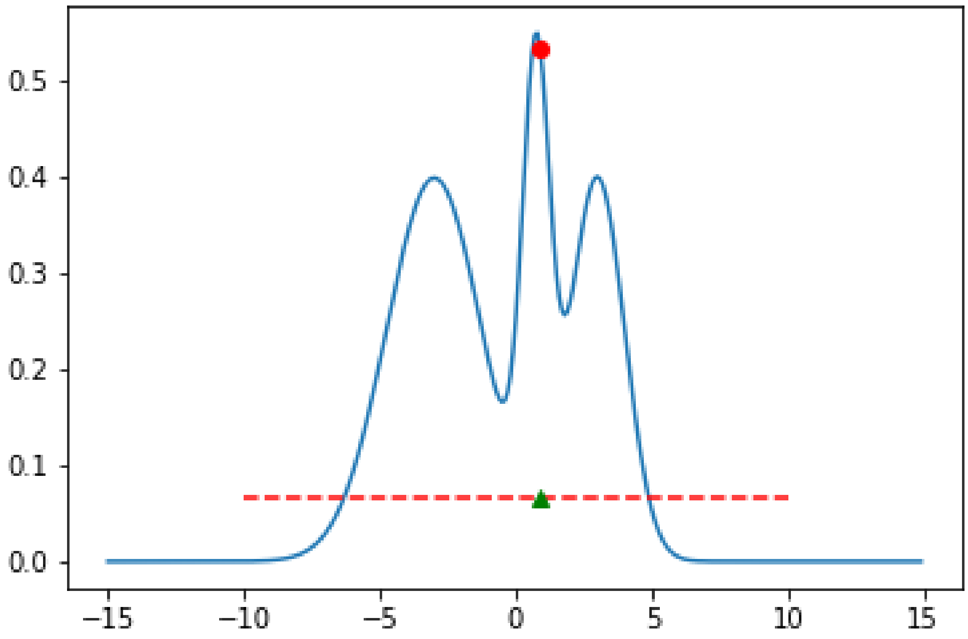
```
0.06719585759020334
```

1.0.6 Define horizontal slice using y

```
In [273]: def __horizontal(sampled_y):
              _x = np.linspace(-10, 10, 50)
              horizontal_line = np.array([sampled_y for i in range(len(_x))])
              return _x, horizontal_line

          current_horizontal = __horizontal(current_sampled_y)

          plt.plot(X, Y)
          plt.plot(current_horizontal[0], current_horizontal[1], 'r--')
          plt.plot([current_x], [current_fx], 'ro')
          plt.plot([current_x], [current_sampled_y], 'g^')
          plt.show()
```



1.0.7 Define function for estimating the sampling interval using doubling update

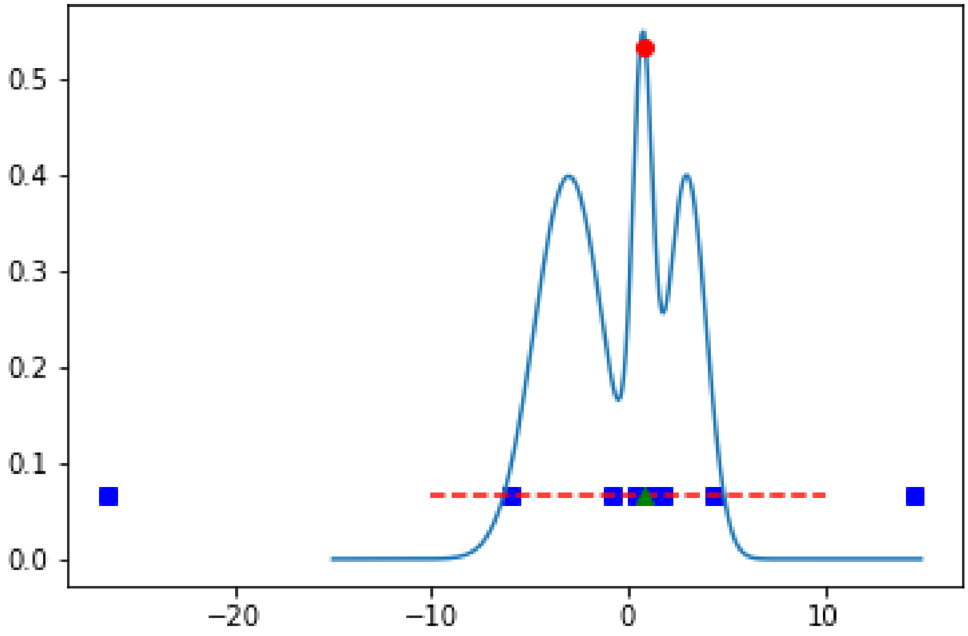
```
In [275]: def __doubling_update(x, y, sampled_y, w=0.01, p=100):
              even = []
              odd = []
              r = x + w
              l = x
              k = p

              while k > 0 and (sampled_y < __fx(l) or sampled_y < __fx(r)):
                  w = 2*w
                  k = k - 1
                  if k % 2 == 0:
                      l = r - w
                      even.append(l)
                  else:
                      r = l + w
                      odd.append(r)

              patches = np.concatenate((even,odd), axis=None)
              return l, r, patches

          double = __doubling_update(current_x, current_fx, current_sampled_y)
```

```
ascisse = np.array([current_sampled_y for i in range(12)])
plt.plot(double[2], ascisse, 'bs')
plt.plot(X, Y)
plt.plot(current_horizontal[0], current_horizontal[1], 'r--')
plt.plot([current_x], [current_fx], 'ro')
plt.plot([current_x], [current_sampled_y], 'g^')
plt.show()
```



1.0.8 Define updating rule for sampling x_n from new interval

```
In [276]: def update_x_double(current_sampled_y):
              curr_x = __sample_x(double[0], double[1])
              while current_sampled_y > __fx(curr_x):
                  curr_x = __sample_x(double[0], double[1])
              return curr_x

          updated_x_double = update_x_double(current_sampled_y)
          print(updated_x_double)
```

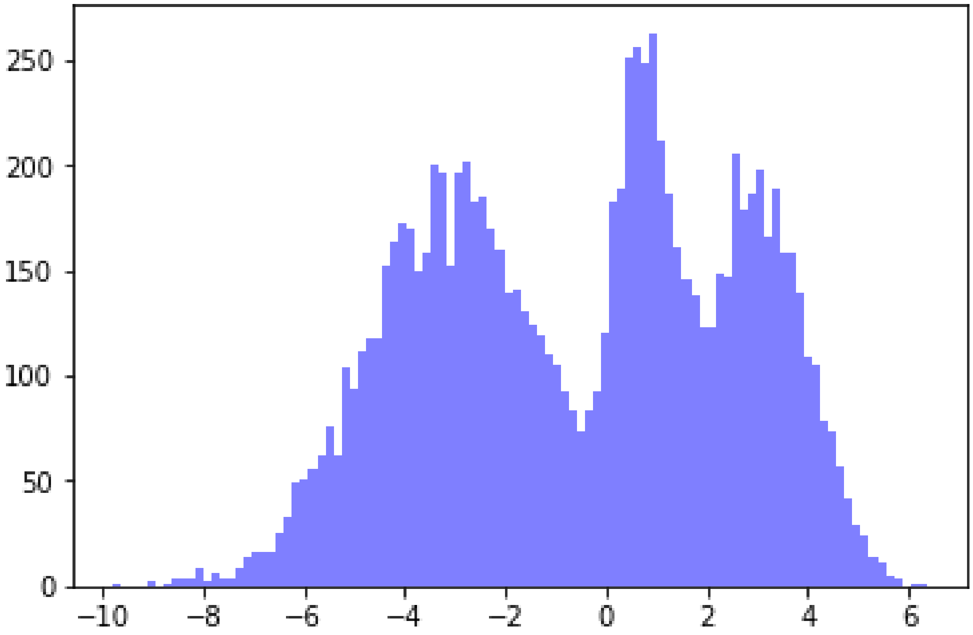
```
0.8849872957049207
```

1.0.9 Sample and plot results

```
In [277]: samples = []
          i = 0
```

```
while i <10000:
    new_x = update_x_double(current_sampled_y)
    new_fx = __fx(new_x)
    new_sampled_y = __sample_y(new_x, new_fx)
    new_double = __doubling_update(new_x, new_fx, new_sampled_y)
    samples.append(round(new_x,2))
    i = i+1
    current_sampled_y = new_sampled_y

num_bins = 100
n, bins, patches = plt.hist(samples, num_bins, facecolor='blue', alpha=0.5)
plt.show()
```



1.0.10 Define Stepout Update function

```
In [279]: def __stepout_update(x, y, sampled_y, w=2):
              patches = []
              r = x
              l = x

              while sampled_y < __fx(l):
                  l = l - w
                  patches.append(l)

              while sampled_y < __fx(r):
```

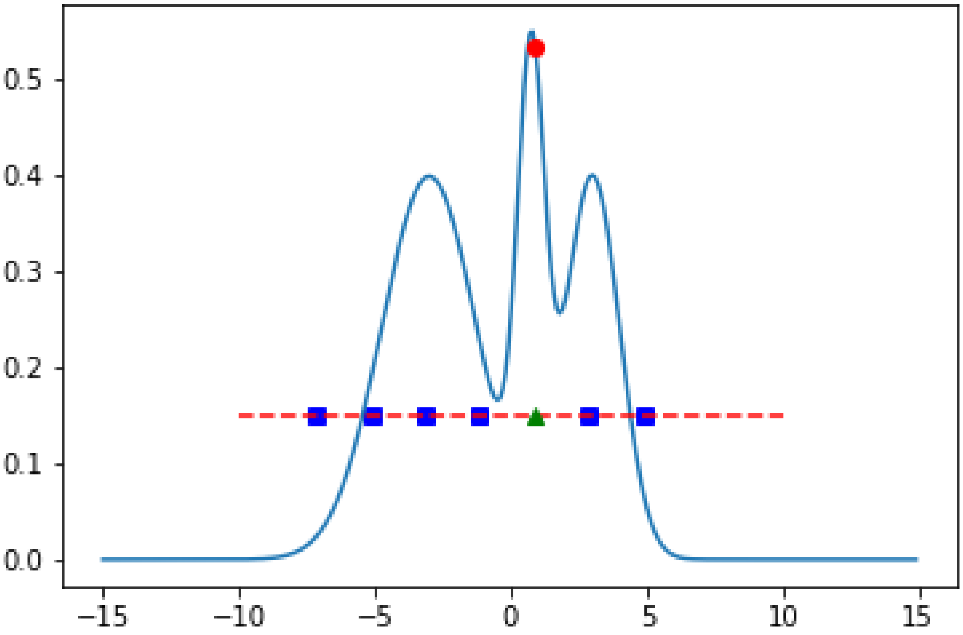
```
        r = r + w
        patches.append(r)

    return l, r, patches

stepout = __stepout_update(current_x, current_fx, current_sampled_y)

ascisse = np.array([current_sampled_y for i in range(6)])
current_horizontal = __horizontal(current_sampled_y)


plt.plot(stepout[2], ascisse, 'bs')
plt.plot(X, Y)
plt.plot(current_horizontal[0], current_horizontal[1], 'r--')
plt.plot([current_x], [current_fx], 'ro')
plt.plot([current_x], [current_sampled_y], 'g^')
plt.show()
print(current_sampled_y)
```



```
0.14919827089334547
```

1.0.11 Define updating rule for sampling x_n from new interval

```
In [280]: def update_x_stepout(current_sampled_y):
              curr_x = __sample_x(stepout[0], stepout[1])
```

```
            while current_sampled_y > __fx(curr_x):
                curr_x = __sample_x(stepout[0], stepout[1])
            return curr_x

        updated_x_stepout = update_x_stepout(current_sampled_y)
        print(updated_x)
```

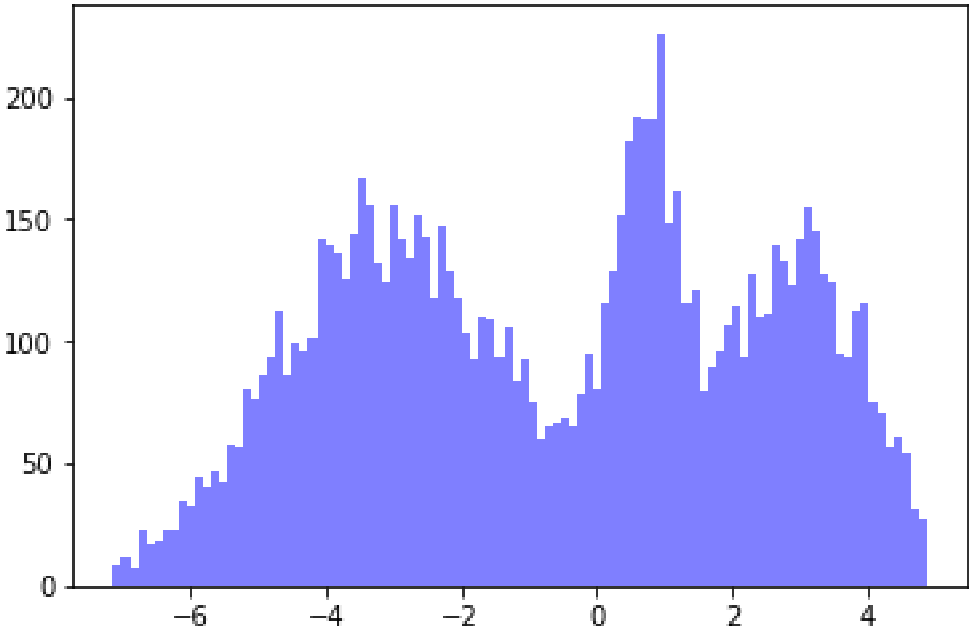
```
0.5987416936679817
```

1.0.12 Sample and plot results

```
In [281]: samples = []
          i = 0

          while i <10000:
              new_x = update_x_stepout(current_sampled_y)
              new_fx = __fx(new_x)
              new_sampled_y = __sample_y(new_x, new_fx)
              new_horizontal = __horizontal(new_sampled_y)
              new_double = __stepout_update(new_x, new_fx, new_sampled_y)
              samples.append(round(new_x,2))
              i = i+1
              current_sampled_y = new_sampled_y

          num_bins = 100
          n, bins, patches = plt.hist(samples, num_bins, facecolor='blue', alpha=0.5)
          plt.show()
```


200
150
100
50
0
−6
−4
−2
0
2
4