In [6]:
```
%pylab inline
```

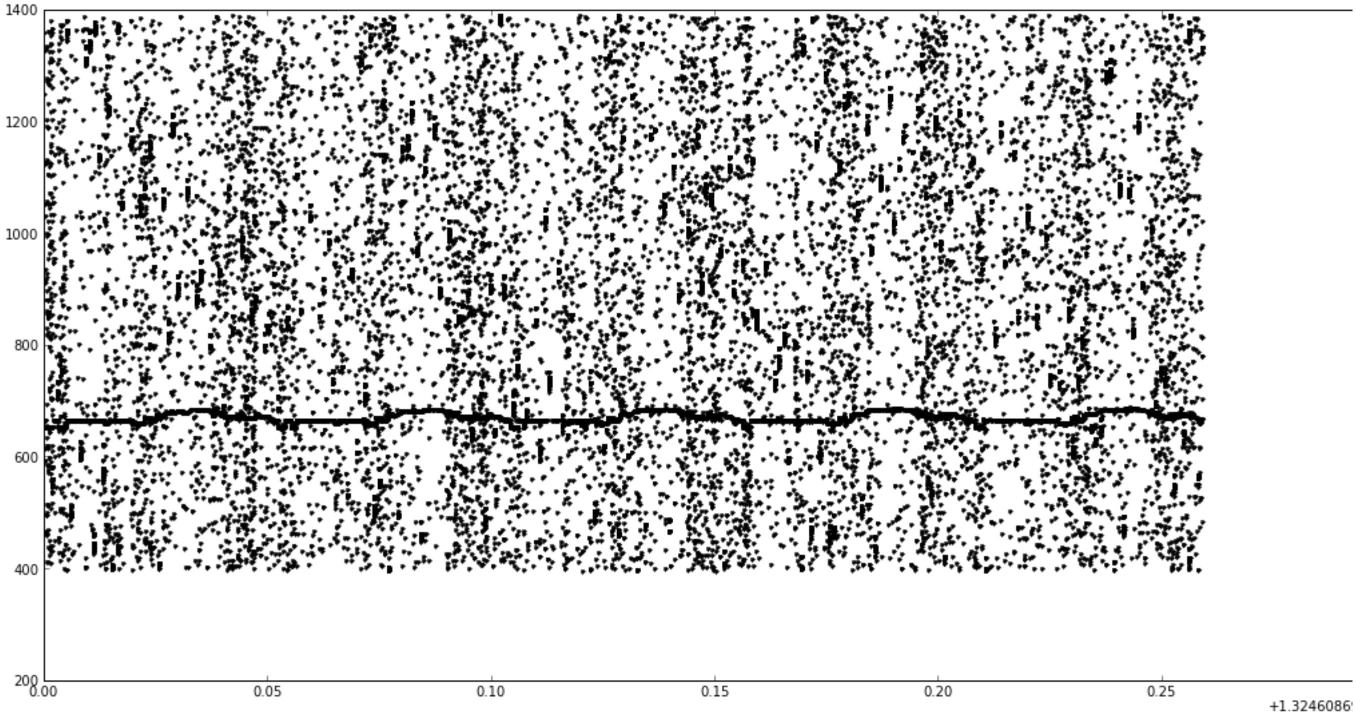
```
Welcome to pylab, a matplotlib-based Python environment [backend: module://IPython.zmq.pylab.backend_inline].
For more information, type 'help(pylab)'.
```

In [38]:
```
# TODO: is this the best way to handle constants like this?
T_IDX = 0
Z_IDX = 3
L_IDX = 5 # label; 1 for surface, 0 for noise

# appends a 6th field to the photon; if it is flipped to 0, that indicates
# that the photon has been labeled as some form of noise
def read_line(line):
    ts, x, y, z, ch = line.split()
    return [float(ts), float(x), float(y), float(z), int(ch), 1]

def transpose(m): return zip(*m)

def read_file(filename):
    return map(read_line, open(filename))
```

In [39]:
```
mcmurdo_filename = 'Presentations/Generators/data/mcmurdo.txyzs'
mcmurdo = read_file(mcmurdo_filename)
t,x,y,z,x,l = transpose(mcmurdo)
```

In [40]:
```
figsize(18, 9)
plot(t, z, 'k.')#, markersize=0.5, markerfacecolor='black')
```

Out[40]:
```
[<matplotlib.lines.Line2D at 0x104d301d0>]
```



In [41]:
```
def stream_photons(filename):
    for line in open(filename):
        yield read_line(line)
```

In [83]:
```
photons = stream_photons(mcmurdo_filename)
```

In [50]:
```
next(photons)
```

Out[50]:
```
[1324608692.000074, -110.135, -23.918, 880.212, 42, 1]
```

In [42]:
```
# This yields pulses, which are a list of all photons with the same timestamp
# Example Usage:
# pulses = stream_pulses(stream_photons('ICP4.F23.0d.mcmurdo1.txyzs'))
def stream_pulses(photon_stream):
    group = []
    for photon in photon_stream:
        if group != [] and photon[T_IDX] != group[0][T_IDX]:
            yield group
            group = []
        group.append(photon)
```

In [53]:
```
pulses = stream_pulses(stream_photons(mcmurdo_filename))
```

In [62]:
```
next(pulses)
```

Out[62]:
```
[[1324608692.000428, -113.742, -30.385, 910.47, 10, 1],
 [1324608692.000428, -81.591, -21.993, 654.398, 18, 1],
 [1324608692.000428, -81.591, -22.148, 654.381, 22, 1],
 [1324608692.000428, -81.263, -21.522, 654.418, 31, 1],
 [1324608692.000428, -113.325, -29.911, 916.313, 43, 1]]
```

In [43]:
```
# This yields overlapping swaths that advance by one pulse at a time
# and contain 2*swath_radius+1 pulses
def stream_swaths(pulse_stream, swath_radius):
    swath = []
    for pulse in pulse_stream:
        if len(swath) < 2*swath_radius + 1:
            swath.append(pulse)
        else:  # only want to run this if we have a full swath
            yield swath
            swath = swath[1:]   # remove first pulse
            swath.append(pulse) # add new pulse to the end
```

In [78]:
```
swaths = stream_swaths(stream_pulses(stream_photons(mcmurdo_filename)), 10)
```

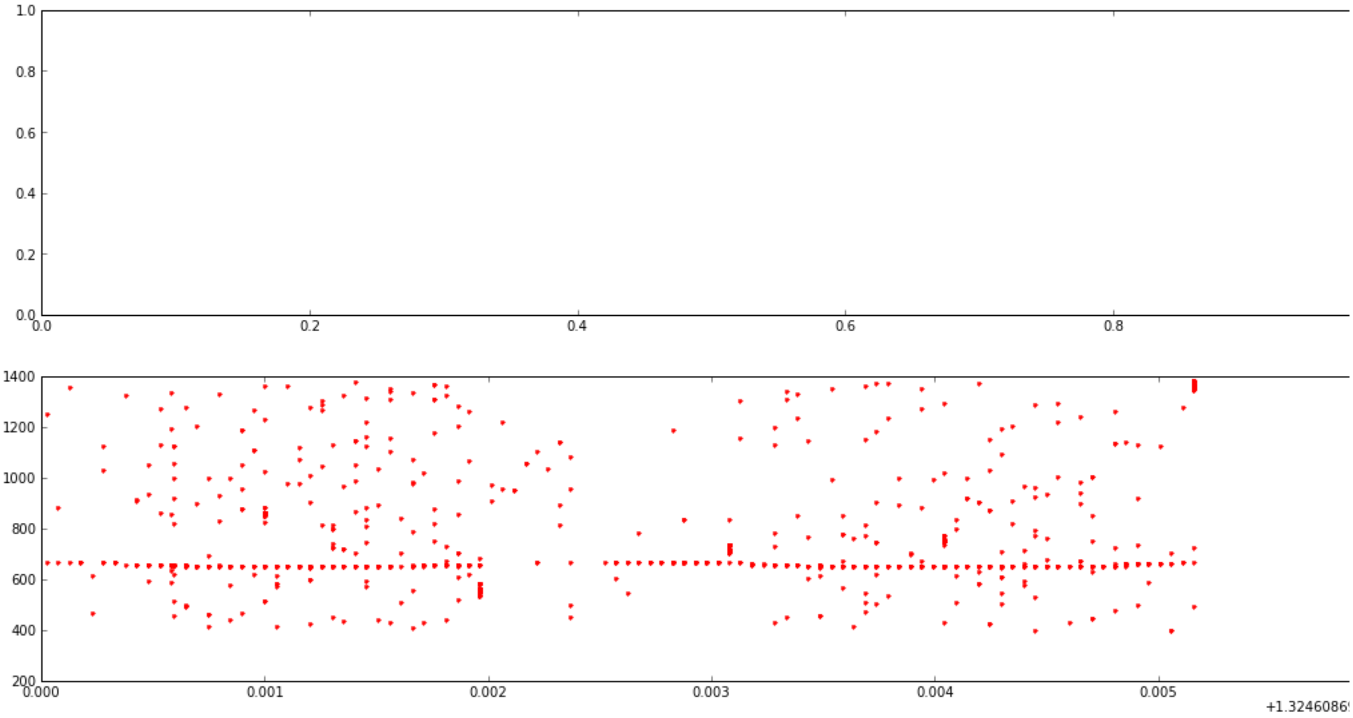
In [79]:
```
next(swaths)
```

```
  [1324608692.000694, -80.605, -24.615, 653.421, 4, 1],
  [1324608692.000694, -80.256, -24.759, 653.211, 6, 1],
  [1324608692.000694, -110.411, -34.208, 896.8, 14, 1],
  [1324608692.000694, -147.703, -45.28, 1202.196, 29, 1],
  [1324608692.000694, -80.196, -24.686, 653.389, 32, 1],
  [1324608692.000694, -80.217, -24.382, 653.598, 34, 1],
  [1324608692.000694, -80.121, -24.299, 653.485, 39, 1],
  [1324608692.000694, -80.122, -24.299, 653.492, 39, 1],
  [1324608692.000694, -79.98, -24.458, 653.636, 45, 1]],
 [[1324608692.000746, -56.732, -17.667, 461.253, 1, 1],
  [1324608692.000746, -56.732, -17.666, 461.248, 1, 1],
  [1324608692.000746, -85.649, -26.785, 694.933, 7, 1],
  [1324608692.000746, -80.338, -25.328, 653.143, 15, 1],
  [1324608692.000746, -80.114, -25.408, 653.287, 20, 1],
  [1324608692.000746, -122.664, -38.903, 1000.263, 20, 1],
  [1324608692.000746, -80.214, -24.872, 653.507, 30, 1],
  [1324608692.000746, -80.108, -24.786, 653.312, 33, 1],
  [1324608692.000746, -50.484, -15.714, 412.965, 44, 1]],
 [[1324608692.000797, -80.264, -25.587, 653.178, 4, 1],
  [1324608692.000797, -113.668, -36.597, 928.745, 6, 1],
  [1324608692.000797, -80.41, -25.739, 653.033, 10, 1],
```

In [93]:
```
# This plots a set number of photons, color-coded by label
# so far, I'm only using this in interactive mode, but it's really useful...
#import matplotlib.pyplot as plt
def plot_photons(density_stream, plot_style='r.', num_pulses=0):
    # TODO: this is silly - I can directly use treat density_stream in the
    #       list comprehension for surf/noise construction. But - how to
    #       stop that one after num_pulses? Maybe use int,elem = enumerate(list)?
    photons = []
    if num_pulses > 0:
        for ii in range(num_pulses):
            photons.extend(next(density_stream))
    else:
        for elem in density_stream:
            photons.extend(elem)
    # TODO: is there a better way to split the data than having two separate
    #        list comprehensions? Norvig said he didn't know anything better for this.
    surf = transpose([photon for photon in photons if photon[L_IDX] == 1])
    noise = transpose([photon for photon in photons if photon[L_IDX] == 0])

    fig = plt.figure()
    ax = fig.add_subplot(2,1,1)

    if noise:
        ax.plot(noise[T_IDX], noise[Z_IDX], 'k.')
    ax = fig.add_subplot(2,1,2)
    if surf:
        ax.plot(surf[T_IDX], surf[Z_IDX], plot_style)
```

In [116]:
```
def filter_flatten(swath):
    pulses = []
    for pulse in swath:
        pulses.append(pulse)
    return pulses
```

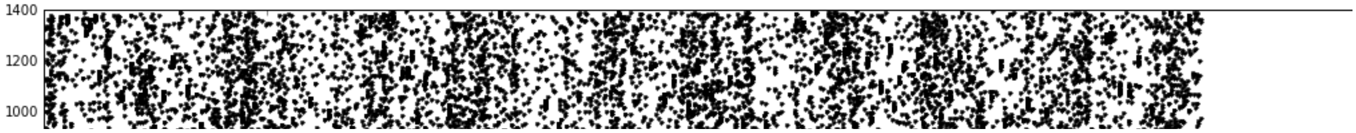
In [99]:
```
swaths = stream_swaths(stream_pulses(stream_photons(mcmurdo_filename)), 50)
photons = filter_flatten(next(swaths))
plot_photons(photons)
```

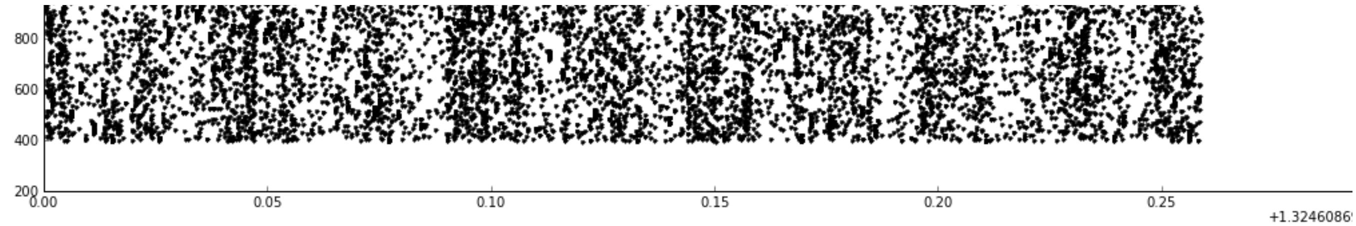


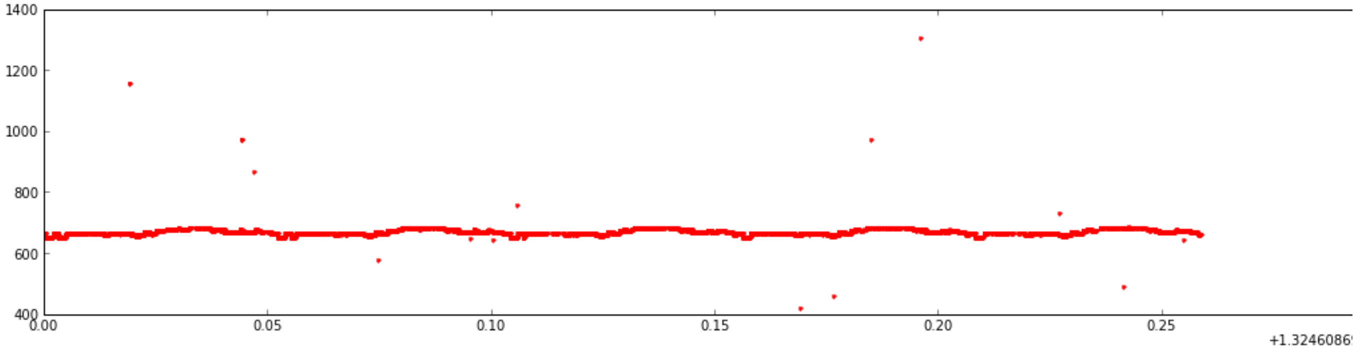
In [104]:
```
# This yields one pulse worth of photons, annotated with the local
# density (number of other photons in the pulse that are within
# neighbor_thresh of their z)
# If density_threshold is set, also require that the last field of the
# photon is greater than the density_threshold
def filter_solar(swath_stream, neighbor_threshold, density_threshold):
    for swath in swath_stream:
        l = len(swath)
        center_idx = (l-1)/2
        center_pulse = swath[center_idx]
        other_pulses = [pulse for idx,pulse in enumerate(swath) if idx!=center_idx]
        other_photons = [photon for sublist in other_pulses for photon in sublist]
        # TODO: make my own count function to make this more readable ... (possibly count = sum)
        for photon in center_pulse:
            density = len([1 for other in other_photons if
                               abs(photon[Z_IDX]-other[Z_IDX]) < neighbor_threshold and
                               other[L_IDX] != 0])
            if density < density_threshold:
                photon[L_IDX] = 0
        yield center_pulse
```

In [110]:
```
def one_pass(filename):
    pulses = stream_pulses(stream_photons(filename))
    density_stream = filter_solar(stream_swaths(pulses, 5), 1.0, 3)
    densities = [density for density in density_stream]
    plot_photons(densities)
```

In [111]:
```
one_pass(mcmurdo_filename)
```

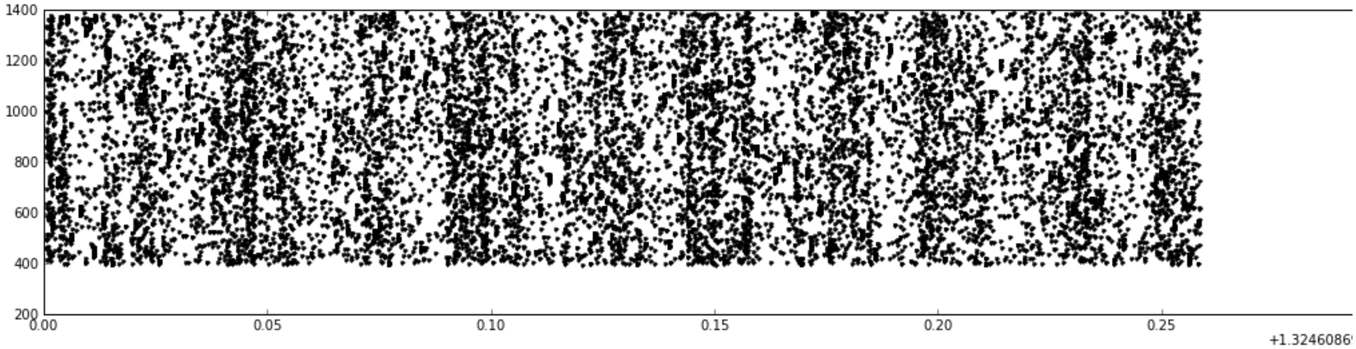


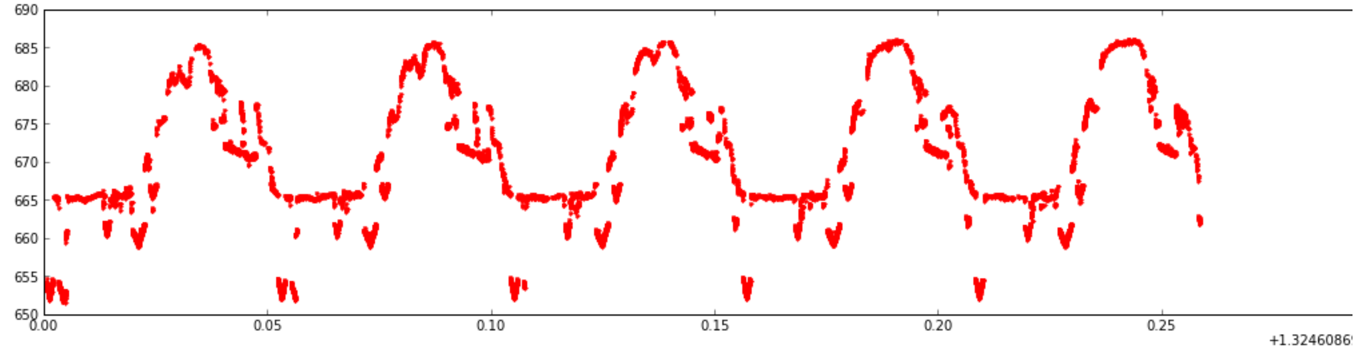




```
In [114]: def two_pass(filename):
              pulses = stream_pulses(stream_photons(filename))
              density_stream = filter_solar(stream_swaths(pulses, 5), 1.0, 3)
              density_stream2 = filter_solar(stream_swaths(density_stream, 5), 1.0, 3)
              densities = [density for density in density_stream2]
              plot_photons(densities)
```

```
In [115]: two_pass(mcmurdo_filename)
```





```
In [ ]:
```