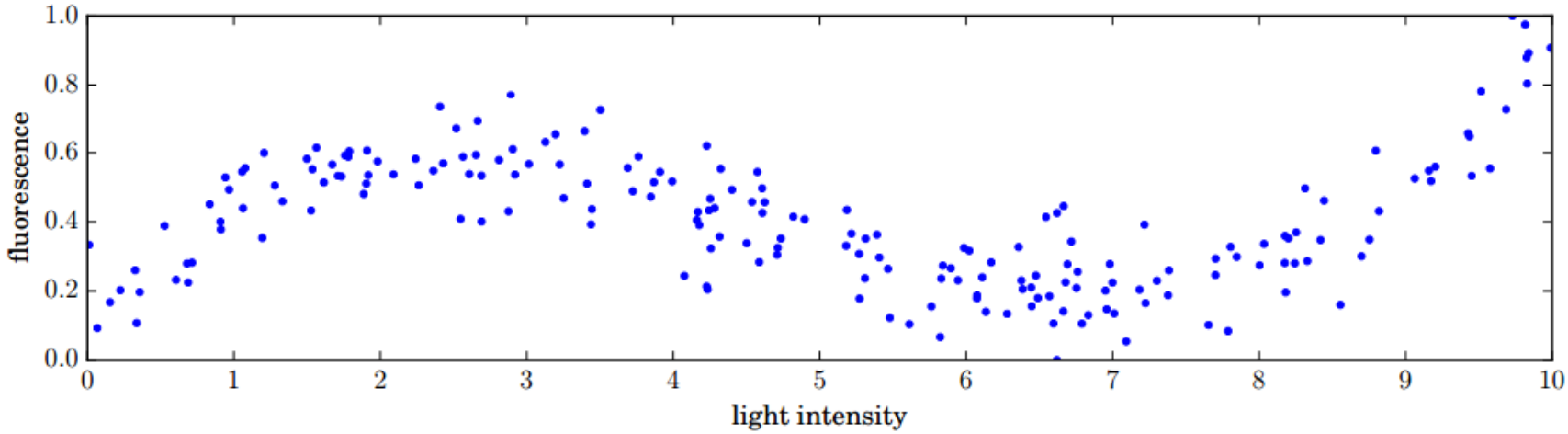
3. Spline fitting [10 pts]

by Roumen Guha, on Sunday, February 26th, 2017

We are running a series of experiments to evaluate the properties of a new fluorescent material. As we vary the intensity of the incident light, the material should fluoresce different amounts. Unfortunately, the material isn't perfectly uniform and our method for measuring fluorescence is not very accurate. After testing 200 different intensities, we obtained the result below (also available in **xy_data.csv**). The intensities x_i and fluorescences y_i are recorded in the first and second columns of the data matrix, respectively.

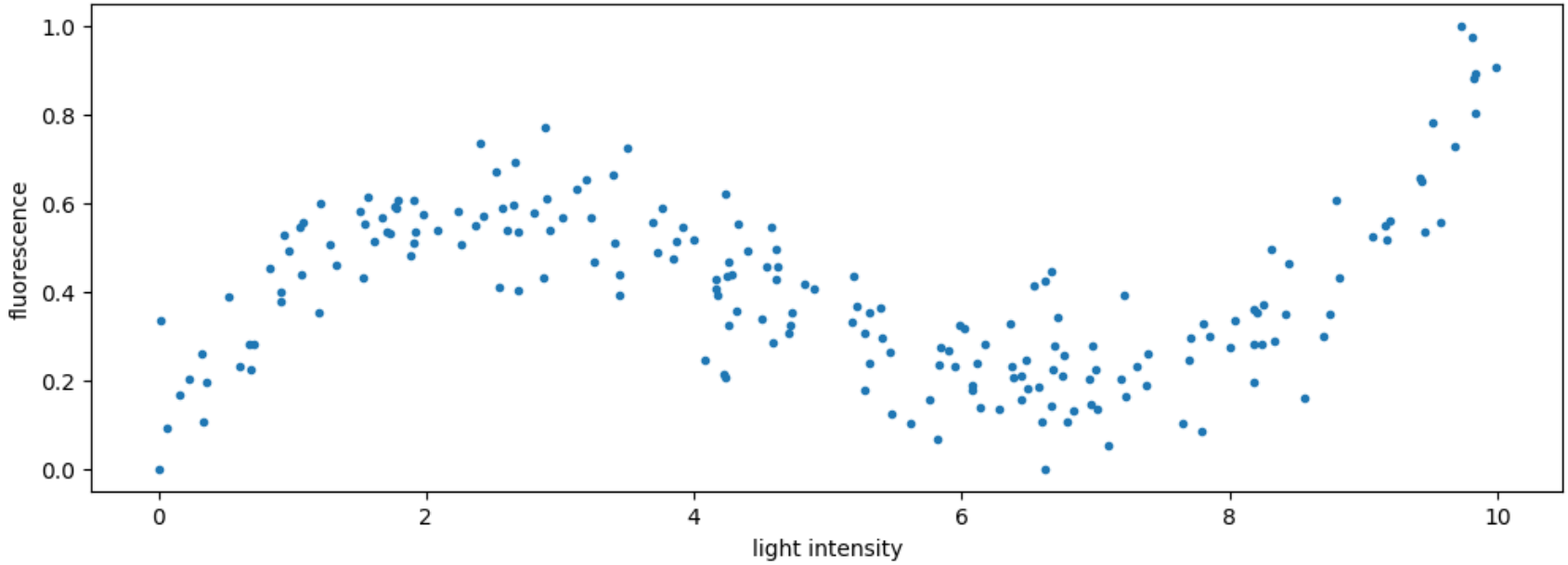


The material has interesting nonlinear properties, and we would like to characterize the relationship between intensity and fluorescence by using an approximate model that agrees well with the trend of our experimental data. Although there is noise in the data, we know from physics that ***the fluorescence must be zero when the intensity is zero.*** This fact must be reflected in all of our models!

In [3]:
```
raw = readcsv("xy_data - modified.csv") # modified original file to include 0 fluorescence when light intensity
x = raw[:,1]; # light intensity
y = raw[:,2]; # fluorescence

using PyPlot
figure(figsize=(12,4))
plot(x,y, ".")
xlabel("light intensity")
ylabel("fluorescence")

;
```



a) Polynomial fit. Find the best cubic polynomial fit to the data. In other words, look for a function of the form

$$y = a_1 x^3 + a_2 x^2 + a_3 x + a4$$

that has the best possible agreement with the data. Remember that the model should have zero fluorescence when the intensity is zero! Include a plot of the data along with your best-fit cubic on the same axes.

In [4]:
```
# order of polynomial to use
k = 3

# fit using a function of the form f(x) = u1 x^k + u2 x^(k-1) + ... + uk x + u{k+1}
n = length(x)
A = zeros(n,k+1)
for i = 1:n
    for j = 1:k+1
        A[i,j] = x[i]^(k+1-j)
    end
end
```

In [5]:
```
using JuMP, Gurobi, Mosek

m = Model(solver=MosekSolver(LOG=0))
#m = Model(solver=GurobiSolver(OutputFlag=0))

@variable(m, u[1:k+1])
@constraint(m, u[4] == 0)
@objective(m, Min, sum( (y - A*u).^2 ) )
status = solve(m)
uopt = getvalue(u)
println(status)
```

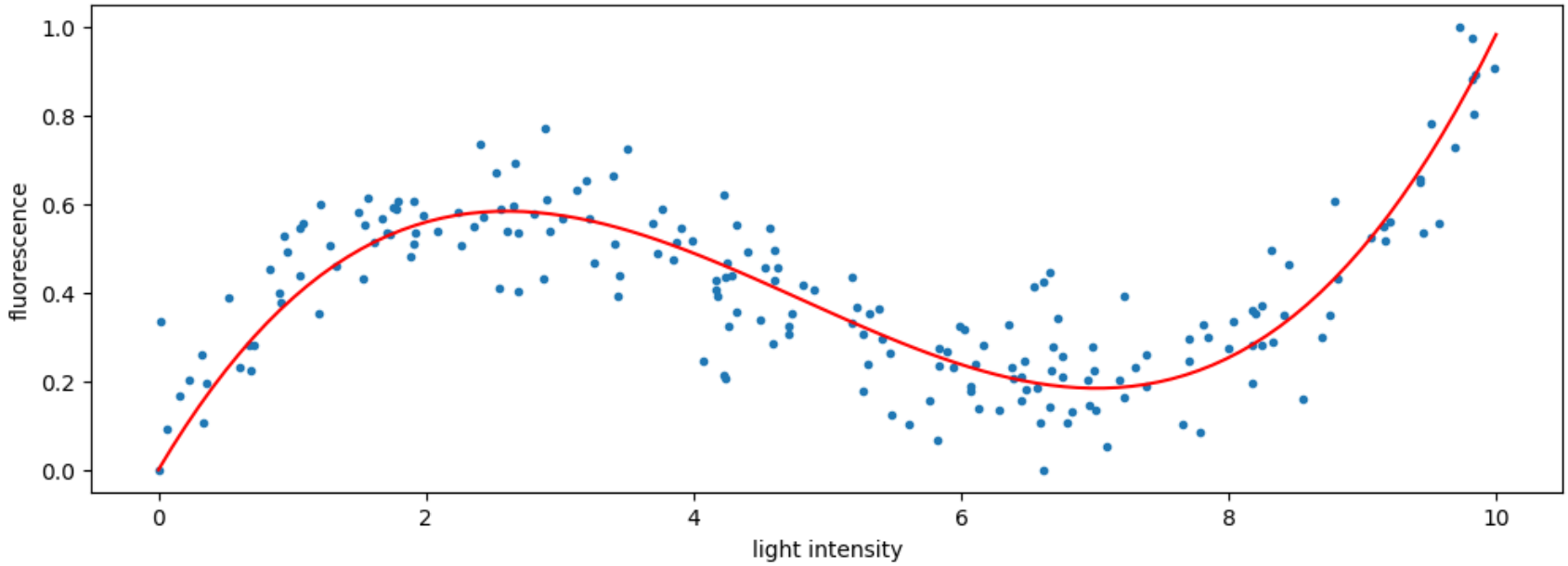
```
Optimal
```

In [6]:
```
using PyPlot

npts = 100
xfine = linspace(0,10,npts)
ffine = ones(npts)
for j = 1:k
    ffine = [ffine.*xfine ones(npts)]
end
yfine = ffine * uopt

figure(figsize=(12,4))
plot( x, y, ".")
plot( xfine, yfine, "r-")
xlabel("light intensity")
ylabel("fluorescence")

;
```



b) Spline fit. Instead of using a single cubic polynomial, we will look for a fit to the data using two quadratic polynomials. Specifically, we want to find coefficients p_i and q_i so that our data is well modeled by the piecewise quadratic function:

$$y = \begin{cases} p_1 x^2 + p_2 x + p_3 & \text{if } 0 \leq x < 4 \\ q_1 x^2 + q_2 x + q_3 & \text{if } 4 \leq x < 10 \end{cases}$$

These quadratic functions must be designed so that:

- as in the cubic model, there is zero fluorescence when the intensity is zero.
- both quadratic pieces have the same value at x = 4.
- both quadratic pieces have the same slope at x = 4.

In other words, we are looking for a *smooth* piecewise quadratic. This is also known as a *spline* (this is just one type of spline, there are many other types!). Include a plot of the data along with your best-fit model.

In [7]:
```
k = 2

# fit using a function of the form f(x) = u1 x^k + u2 x^(k-1) + ... + uk x + u{k+1}
n1 = 78 # last index before x[78] which contains a value > 4.0
A = zeros(n1,k+1)
for i = 1:n1
    for j = 1:k+1
        A[i,j] = x[i]^(k+1-j)
    end
end
```

In [8]:
```
using JuMP, Gurobi, Mosek

m1 = Model(solver=MosekSolver(LOG=0))
#m = Model(solver=GurobiSolver(OutputFlag=0))

@variable(m1, u1[1:k+1])
@constraint(m1, u1[3] == 0)
@objective(m1, Min, sum( (y[1:n1] - A*u1).^2 ) )
status = solve(m1)
uopt1 = getvalue(u1)
println(status)

npts = 100
xfine1 = linspace(0,4,npts)
ffine1 = ones(npts)
for j = 1:k
    ffine1 = [ffine1.*xfine1 ones(npts)]
end
yfine1 = ffine1 * uopt1;
```

```
Optimal
```

In [9]:
```
k = 2

# fit using a function of the form f(x) = u1 x^k + u2 x^(k-1) + ... + uk x + u{k+1}
n2 = 78
A = zeros(length(x) - n1, k+1)
for i = 1:length(x)-n1
    for j = 1:k+1
        A[i,j] = x[i+n1]^(k+1-j)
    end
end
```

In [14]:
```
using JuMP, Gurobi, Mosek

m2 = Model(solver=MosekSolver(LOG=0))
#m = Model(solver=GurobiSolver(OutputFlag=0))

@variable(m2, u2[1:k+1])

@objective(m2, Min, sum( (y[n2 + 1:length(x)] - A*u2).^2 ))
status = solve(m2)
uopt2 = getvalue(u2)
println(status)

npts = 100
xfine2 = linspace(4,10,npts)
ffine2 = ones(npts)
for j = 1:k
    ffine2 = [ffine2.*xfine2 ones(npts)]
end
yfine2 = ffine2 * uopt2;
```

```
Optimal
```

In [19]:
```
using PyPlot

figure(figsize=(12,4))
plot( x, y, ".")
plot( xfine1, yfine1, "r-")
plot( xfine2, yfine2-(yfine2[1] - yfine1[end]), "b-")
xlabel("light intensity")
ylabel("fluorescence")

;
```