PS5 Solution

This problem is about predicting the world population. Here is a julia notebook. You can convert to any language. Form the matrix of data for the world population from `http://en.wikipedia.org/wiki/World_population`, table "Estimated world and regional populations at various dates (in millions)" by taking the first two columns from 1950 until 2010.

In[1]:
```
A=[1950 2519
   1955 2756
   1960 2982
   1965 3335
   1970 3692
   1975 4068
   1980 4435
   1985 4831
   1990 5263
   1995 5674
   2000 6070
   2005 6454
   2010 6972];
```

Column 1 of $A$ is time $t$ and column 2, population $P$.

In[2]:
```
t=A[:,1]; P=A[:,2];
```

In[3]:
```
function B(k)
 z=zeros(Int,13,k+1)
 for i=0:k z[:,i+1]=t.^i end
 z
end
```

Now approximate the population in the least-squares sense with the function $a * \exp(b * t)$, and with polynomials of second (quadratic), third (cubic) and fourth (quartic) degree. Plot all aproximations and the original data. Predict the current population by each approximation and compare to the current population from `http://www.worldometers.info/world-population/` (Looks like 7266 so far this year on October 10, 2014 )

Suppose we say 7266 + 63*(80/365) is the actual number this year – you might be able to be a bit more accurate.

In[4]:
```
actual = 7266 + 63*(80/365)
```

Out[4]:
```
7279.808219178082
```

For each time $t$, the quadratic approximation computes a column with first entry $t^0 = 1$, second entry $t^1$ and third entry $t^2$.

In[5]:
```
B(2) # Quadratic
```

Out[5]:
```
13x3 Array{Int64,2}:
 1  1950  3802500
 1  1955  3822025
 1  1960  3841600
 1  1965  3861225
 1  1970  3880900
 1  1975  3900625
 1  1980  3920400
 1  1985  3940225
 1  1990  3960100
 1  1995  3980025
 1  2000  4000000
 1  2005  4020025
 1  2010  4040100
```

Julia uses the backslash to do the least squares computation to get the coefficients $c$.

In[6]:
```
c=B(2)\P compute the coefficient of 1,t, and t^2
```

Out[6]:
```
3-element Array{Float64,1}:
  -0.0714
 -70.6796
   0.0368522
```

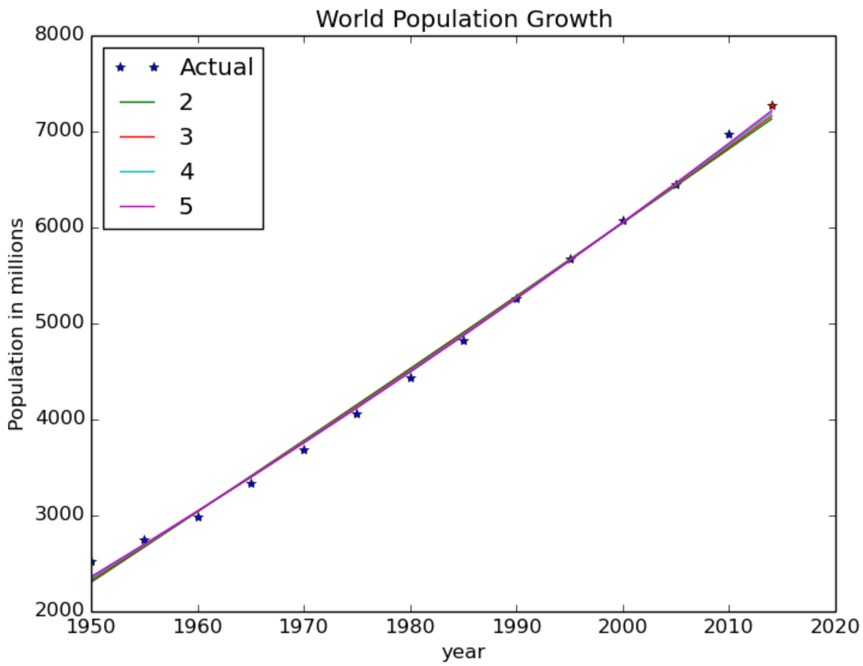
Add a data point for 2014

In[7]:
```
z(k)=[B(k) ;2014.^(0:k)']
```

Up to 5th degree the predictions keep getting a little better

In[8]:
```
nmax=5
plot(t,P,"*")
for n=2:nmax
    c=B(n)\P
    plot([t;2014],z(n)*c)
    println("degree $(n): 2014 Estimate: $(round(z(n)*c)[end])  Actual: $(round(actual))")
end
plot(2014,actual,"r*")
xlabel("year")
ylabel("Population in millions")
title("World Population Growth")
legend({"Actual", (2:nmax)...},loc="upper left")
#savefig("PopulationGrowth")
```

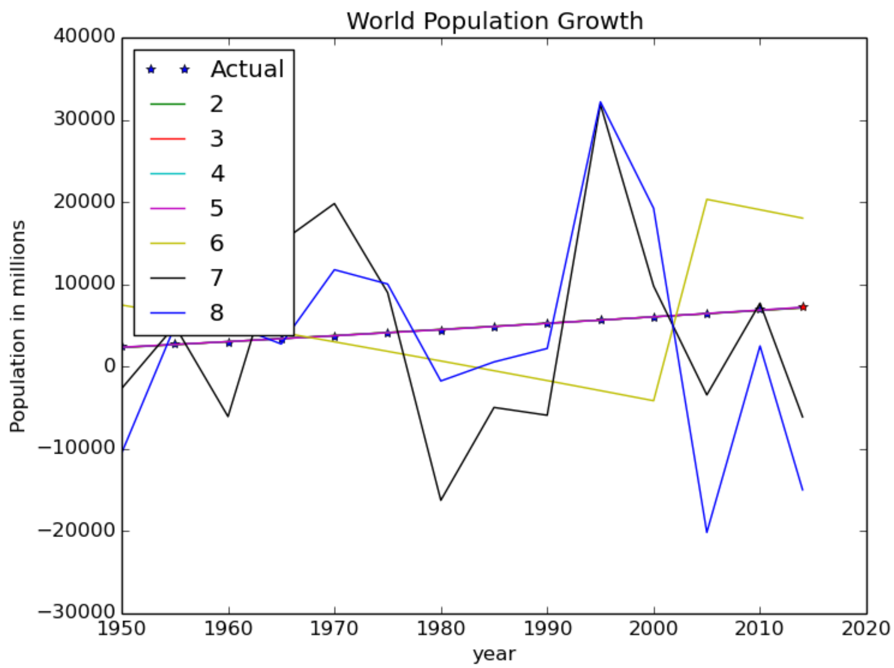
Out[8]:
```
degree 2: 2014 Estimate: 7131.0  Actual: 7280.0
degree 3: 2014 Estimate: 7160.0  Actual: 7280.0
degree 4: 2014 Estimate: 7188.0  Actual: 7280.0
degree 5: 2014 Estimate: 7216.0  Actual: 7280.0
```




After that, the polynomials start going wild – showing that higher degree doesn't necessarily mean a better fit.

In[9]:
```
nmax=8
plot(t,P,"*")
for n=2:nmax
    c=B(n)\P
    plot([t;2014],z(n)*c)
    println("degree $(n): 2014 Estimate: $(round(z(n)*c)[end])  Actual: $(round(actual))")
end
plot(2014,actual,"r*")
xlabel("year")
ylabel("Population in millions")
title("World Population Growth")
legend({"Actual", (2:nmax)...},loc="upper left")
savefig("PopulationGrowth Up to 8th degree")
```

```
Out[9]: degree 2: 2014 Estimate: 7131.0  Actual: 7280.0
        degree 3: 2014 Estimate: 7160.0  Actual: 7280.0
        degree 4: 2014 Estimate: 7188.0  Actual: 7280.0
        degree 5: 2014 Estimate: 7216.0  Actual: 7280.0
        degree 6: 2014 Estimate: 18049.0  Actual: 7280.0
        degree 7: 2014 Estimate: -6108.0  Actual: 7280.0
        degree 8: 2014 Estimate: -14968.0  Actual: 7280.0
```




Note the axes and wild oscillations. There was either a problem with the fit or a numerical problem. Now that I think about it, it's probably a numerical problem. This matrix is badly conditioned. We will learn more about that later in the semester.

The cool thing is Julia lets you up the precision. Sorry I didn't realize this would be happening, but

I hope it's not too late to find interesting.

In[10]:
```
with_bigfloat_precision(500) do

actual = BigFloat(7266) + 63*(80/365)

A=BigFloat[1950 2519
   1955 2756
   1960 2982
   1965 3335
   1970 3692
   1975 4068
   1980 4435
   1985 4831
   1990 5263
   1995 5674
   2000 6070
   2005 6454
   2010 6972]
t=A[:,1]; P=A[:,2];

function B(k)
 zz=zeros(BigFloat,13,k+1)
 for i=0:k zz[:,i+1]=(t.^i) end
 zz
end

function z(k)
  [B(k) ; BigFloat(2014).^(0:k)']
   end

nmax=8
for n=2:nmax
    c=B(n)\P
    println("degree $(n): 2014 Estimate: $(int(z(n)*c)[end])  Actual: $(int(actual))")
end

end
```

Out[10]:
```
degree 2: 2014 Estimate: 7356  Actual: 7280
degree 3: 2014 Estimate: 7259  Actual: 7280
degree 4: 2014 Estimate: 7353  Actual: 7280
degree 5: 2014 Estimate: 7415  Actual: 7280
degree 6: 2014 Estimate: 7531  Actual: 7280
degree 7: 2014 Estimate: 7863  Actual: 7280
degree 8: 2014 Estimate: 7565  Actual: 7280
```

We see indeed it was a numerical issue. While the data doesn't lend itself to good prediction, we do see that at least it didn't have to go too wild. We didn't expect you to notice this.

We were hoping for better luck with an exponential fit. It was not a good fit. Nonetheless explain how this works:

In[11]:
```
c=[t.^0 t]\log(P)
```

Out[11]:
```
2-element Array{Float64,1}:
 -25.7489
   0.017232
```

In[12]:
```
exp(c[1] + 2014 * c[2] )
```

Out[12]:
```
7756.98309596893
```

What is going on is we are fitting the log of the data to a straight line. The result is that $\log(P) \approx c_1 + tc_2$. Exponentiating we have that $P \approx \exp(c_1 + tc_2)$. Plugging in $t = 2014$ gives the exponential prediction.