Interpolation

Numerical Methods, Lecture 10.

INTERPOLATION

Last time we considered regression, function fitting, where best fitting linear, quadratic, exponential, etc. functions were determined from given knots. Generally the fitted function does not pass through the specified knots but ideally approximates them closely.

In the interpolation problem we have a mathematical formula which perfectly represents the given knots by reproducing measurements exactly at these point and within these knots the formula can be used for approximation. Graph of the function passes through the knots.

INTERPOLATION WITH ONE POLYNOMIAL

A polynomial can be given as

\[ f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 \]

Coefficients \( a_n, a_{n-1}, \cdots, a_1, a_0 \) are reals, \( n \) is a nonnegative integer, the degree of the polynomial. A first-degree polynomial is a linear function, its graph is a straight line, whereas graphs of second and higher degree polynomials are parabolas or curves. The higher the degree, the more 'turns' and inflexion points, hence a more complicated shape this graph may have.

Let us have a data set of \( n \) points. This data set may be approximated by polynomials of increasing degree up to at most of degree \((n-1)\). Fit with a lower degree polynomial is a regression, while fit with polynomial of degree \((n-1)\) is interpolation and this polynomial passes through all the points or knots. This is called polynomial interpolation, a **global** one since one function is fitted to all the data. Let us consider an example.
Interpolation

Numerical Methods, Lecture 10.

Wind turbine output power depends on wind velocity. Experimental data for five different velocities are shown in the following table:

<table>
<thead>
<tr>
<th>Wind velocity [km/h]</th>
<th>22</th>
<th>35</th>
<th>48</th>
<th>61</th>
<th>74</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power [W]</td>
<td>320</td>
<td>490</td>
<td>540</td>
<td>500</td>
<td>480</td>
</tr>
</tbody>
</table>

A fourth degree polynomial can be fitted to these five data points. Let us interpolate wind turbine output power for wind velocities 42 and 68 km/h. What wind velocity produces 400 W output power? For the computation load the data file windturbine.txt.

```matlab
% wind turbine data
clear all; close all; clc;
data = load('windturbine.txt')
v = data(:,1) % wind velocity
p = data(:,2) % output power
figure(1)
plot(v,p,'r*')

% compute coefficients
a = polyfit(v,p,4)
% 0.0001  -0.0171   0.5627  12.0190  -62.0517

% evaluate polynomial at desired points
polyval(a,42) % 531.7853
polyval(a,68) % 476.5008

% Alternatively define polynomial function for easy plotting:
fp = @(x) polyval(a,x)
hold on;
hold on;
hold on;
hold on;
hold on;
ezplot(fp,[min(v) max(v)])
```

Coefficients \( a_4, a_3, \ldots, a_1, a_0 \) of the fourth degree interpolation polynomial may be computed by solving a system of five linear equations specified for the five points, just like we learned in case of regression. Matrix \( A \) of the system is called Vandermonde matrix.

\[
\begin{align*}
y_1 &= a_4 x_1^4 + a_3 x_1^3 + a_2 x_1^2 + a_1 x_1 + a_0 \\
y_2 &= a_4 x_2^4 + a_3 x_2^3 + a_2 x_2^2 + a_1 x_2 + a_0 \\
y_3 &= a_4 x_3^4 + a_3 x_3^3 + a_2 x_3^2 + a_1 x_3 + a_0 \\
y_4 &= a_4 x_4^4 + a_3 x_4^3 + a_2 x_4^2 + a_1 x_4 + a_0 \\
y_5 &= a_4 x_5^4 + a_3 x_5^3 + a_2 x_5^2 + a_1 x_5 + a_0
\end{align*}
\]

\[
\begin{align*}
\begin{pmatrix}
x_1^4 & x_1^3 & x_1^2 & x_1 & 1 \\
x_2^4 & x_2^3 & x_2^2 & x_2 & 1 \\
x_3^4 & x_3^3 & x_3^2 & x_3 & 1 \\
x_4^4 & x_4^3 & x_4^2 & x_4 & 1 \\
x_5^4 & x_5^3 & x_5^2 & x_5 & 1
\end{pmatrix} \cdot 
\begin{pmatrix}
a_4 \\
a_3 \\
a_2 \\
a_1 \\
a_0
\end{pmatrix} = 
\begin{pmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4 \\
y_5
\end{pmatrix}
\rightarrow A = 
\begin{pmatrix}
x_1^4 & x_1^3 & x_1^2 & x_1 & 1 \\
x_2^4 & x_2^3 & x_2^2 & x_2 & 1 \\
x_3^4 & x_3^3 & x_3^2 & x_3 & 1 \\
x_4^4 & x_4^3 & x_4^2 & x_4 & 1 \\
x_5^4 & x_5^3 & x_5^2 & x_5 & 1
\end{pmatrix} ; b = 
\begin{pmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4 \\
y_5
\end{pmatrix}
\]

Alternatively built-in Matlab functions `polyfit`, `polyval` can be used here as well. To make things simple let us now use these functions.

\[
> a = \text{polyfit}(v,p,4) \ % 0.0001 -0.0171 0.5627 12.0190 -62.0517
\]

We can specify for `polyfit` the degree of the fitted polynomial and it gives us coefficients of the polynomial in decreasing degree:

\[
a_i = 0.0001; a_3 = -0.0171; a_2 = 0.5627; a_1 = 12.0190; a_0 = -62.0517.
\]

The `polyval` function evaluates a polynomial specified by its coefficients for an arbitrary argument, e.g. at the sought place 42 km/h:

\[
> \text{polyval}(a,42) \ % 531.7853
\]

Alternatively we may define the polynomial function for easy plotting:

\[
> \text{fp} = @(x) \text{polyval}(a,x)
> \text{fp}(68) \ % 476.5008
> \text{hold on;}
> \text{ezplot(fp,[min(v) max(v)])}
\]

If we want to compute wind velocity for output power of 400 W we need to solve a nonlinear equation. The following nonlinear equation with zero on right hand side is to be solved:

\[ a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x + a_0 - 400 = 0. \]

The required initial value can be found from the plot of the function.

\[
\begin{align*}
> & \quad h = @(x) \text{fp}(x) - 400 \\
> & \quad x200 = \text{fzero}(h,30) \% 27.1296
\end{align*}
\]

We found that for wind velocity 42 km/h the output power is 532 W, whereas for 68 km/h it is 477 W. The output power 400 W belongs to 27 km/h wind velocity.

To fit a polynomial of degree \( n \), which in standard form is written as

\[ f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 \]

a system of \((n+1)\) linear equations must be solved. To set up the equations coordinates of all available data points have to be substituted into the standard form. In practice it is not advisable to solve this system, especially for higher degree polynomials, since often we get a badly conditioned coefficient matrix. Let us check the condition number of the coefficient matrix of the above example. For this specify the coefficient matrix of the linear system to be solved. We may either follow the way we learned in case of regression or we may use Matlab’s special command to compute the Vandermode matrix. Both give the same results:

\[
\begin{align*}
> & \quad A = \text{vander}(v) \% \text{alternative: Vandermonde matrix} \\
> & \quad \text{cond}(A) \% 1.5378e+09
\end{align*}
\]

This number is already very big (of order \( 10^9 \)) for our fourth degree polynomial fit, even if this degree is not so high. As a check we compute solution of the linear system of equations and make comparison with polyfit.

\[
\begin{align*}
> & \quad x = A \backslash p \% 0.0001; -0.0171; 0.5627; 12.0190; -62.0517
\end{align*}
\]

## LAGRANGE AND NEWTON INTERPOLATING POLYNOMIALS

There is only one interpolating polynomial for a given set of data points, but there are several different formulations. Let us briefly review two other forms besides the general one that are often better, easier to use and do not lead to a badly conditioned system. One such formulation is due to Lagrange, the other is to Newton.

### LAGRANGE INTERPOLATING POLYNOMIAL

This polynomial can be specified directly from the coordinates of points without any computation or solution of a system of equations, as given below, from \( n \) points:

\[
\begin{align*}
f(x) &= \sum_{i=1}^{n} y_i L_i(x) = \sum_{i=1}^{n} y_i \prod_{j=1}^{n} \frac{x-x_j}{x_i-x_j}
\end{align*}
\]

where \( L_i(x) = \prod_{j=1}^{n} \frac{x-x_j}{x_i-x_j} \) are called Lagrange functions. For two points:

\[
\begin{align*}
f(x) &= \frac{(x-x_2)}{(x_1-x_2)} y_1 + \frac{(x-x_1)}{(x_2-x_1)} y_2
\end{align*}
\]

For three points:
Interpolation

Numerical Methods, Lecture 10.

\[ f[x] = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} y_1 + \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} y_2 + \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)} y_3 \]

- It is clear that the interpolation polynomial can be specified without any calculation, from coordinates of data points only.
- It is uneasy to work with the formula since for each interpolation point the whole equation must be written again, unlike the standard form where we need to substitute the coefficients only.
- If a new point is added to the set of points, all Lagrange functions have to be recomputed. This is different from Newton's form where only some new terms must be computed when a new point is added.

NEWTON INTERPOLATING POLYNOMIAL

Newton's polynomial can recursively be written with so called divided differences. The general form of the polynomial is

\[ f(x) = a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n + \cdots + x^n \]

For two points the coefficients are:

\[ a_1 = y_1, a_2 = \frac{y_2 - y_1}{x_2 - x_1} \]

Let us define first degree divided differences as follows:

\[ f'[x, x_1, x_j] = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} \]

This is the slope of the line and for two points is the same as the coefficient \(a_2\).

In case of three points a second degree divided difference \( f'[x_3, x_2, x_1] \) is specified as difference of the two first degree differences divided by \((x_3 - x_1)\). This gives coefficient \(a_3\) (the first two are the same as before).

\[ a_3 = f'[x_3, x_2, x_1] = \frac{f'[x_3, x_2] - f'[x_2, x_1]}{x_3 - x_1} = \frac{y_3 - y_2}{x_3 - x_2} \cdot \frac{y_2 - y_1}{x_2 - x_1} \]

In the same manner in case of four points a third degree divided difference is specified as difference of two second degree divided differences divided by \((x_4 - x_1)\), and so on. Generally a k-th degree divided difference:

\[ f'[x_{i+k}, \cdots, x_j] = \frac{f'[x_{i+k}, \cdots, x_{i+1}] - f'[x_{i+k-1}, \cdots, x_i]}{x_{i+k} - x_i}, \quad (k = 1, 2, \cdots, n) \quad \text{and} \quad (i = 0, \cdots, n-k) \]

- It is clear that in this case the interpolation polynomial can also be specified without any calculation, from coordinates of data points only.
- It is not so inconvenient to work with, since if coefficients \(a_1, a_2, \cdots, a_n\) are already computed these can be reused later for interpolation of any point.

1 for homework

Interpolation

Numerical Methods, Lecture 10.

- If a new point is added to the set of points, we do not need to start from scratch, only a new coefficient have to be calculated. Hence it is easy to add a new point and points do not have to be ordered.

INTERPOLATION OF RESERVOIR WATER HEIGHT-AREA-VOLUME CURVES

Let us consider a problem in hydraulic engineering. A basic problem in planning of water storage reservoirs is the morphological characteristic curve of the reservoir, which gives volume and area variation as a function of water height. We have the following data:

<table>
<thead>
<tr>
<th>Water level H [cm]</th>
<th>Volume V [10^6 m^3]</th>
<th>Area F [km^2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>336</td>
<td>0.16</td>
<td>0.05</td>
</tr>
<tr>
<td>504</td>
<td>0.36</td>
<td>0.09</td>
</tr>
<tr>
<td>714</td>
<td>0.79</td>
<td>0.19</td>
</tr>
<tr>
<td>976</td>
<td>1.73</td>
<td>0.37</td>
</tr>
<tr>
<td>1302</td>
<td>3.31</td>
<td>0.62</td>
</tr>
<tr>
<td>1628</td>
<td>5.83</td>
<td>0.90</td>
</tr>
<tr>
<td>1812</td>
<td>7.72</td>
<td>1.05</td>
</tr>
<tr>
<td>1932</td>
<td>8.98</td>
<td>1.16</td>
</tr>
<tr>
<td>2142</td>
<td>11.50</td>
<td>1.27</td>
</tr>
</tbody>
</table>

Let us make a water height – volume graph as usual by plotting volume on the horizontal, height on the vertical axis. Interpolate volume for water level of 15 m and conversely, water level belonging to 12 million m^3 volume.

Load data from reservoir.txt

```matlab
clc; clear all; close all;
> data = load('reservoir.txt')
> H = data(:,1); % cm
> V = data(:,2); % million m^3
> F = data(:,3); % km^2
> figure(1)
> plot(V,H,'*'); hold on;
> x = data(:,1);
> polyval(a1,x)
> set(g,'Color','r')
```

In the problem of interpolation an \((n-1)\)-degree polynomial can be fitted by using \(n\) points. Apply this rule to the present problem. We have 9 data, hence a polynomial of degree 8 can be fitted to.

```matlab
> n = length(V) % 9
> a1 = polyfit(V,H,n-1)
> p1 = @(x) polyval(a1,x)
> g=ezplot(p1,[0,max(V)])
> set(g,'Color','r')
```
Interpolation

Numerical Methods, Lecture 10.

What has happened? Is this curve acceptable for interpolation? Certainly it is not. This is over-fitting. In this case a high-degree polynomial reproduces data perfectly but there are oscillations in between (especially at the two sides). The interpolation deviates significantly from the trend of data, hence it is unsuitable for interpolation or extrapolation. This wavy, oscillating behavior is called Runge effect. For few data points when degree of the polynomial is low interpolating polynomials are of good use, but for many points and high degree polynomials we must look for another solution.

Incidentally, Matlab issues a warning after running polyfit:

```
Warning: Polynomial is badly conditioned. Add points with distinct X values, reduce the degree of the polynomial, or try centering and scaling as described in HELP POLYFIT.
```

When we ask for the condition number of Vandermonde matrix the answer is a very big number (order of $10^{10}$) which indicates a badly conditioned matrix.

```matlab
> A = vander(V);
> cond(A) % 2.7260e+10
```

**SPLINE INTERPOLATION**

If interpolation is required when there are many data points we get better results by using several polynomials of low degree than by using one with high degree. Each low degree polynomial is valid over a particular interval only between two or more points. Generally these polynomials have the same degree and differ only in their coefficients. When first degree polynomials are used, data points are connected by straight line segments. In case of second degree (quadratic) or third degree (cubic) polynomials data points are connected by curves. This is a polynomial interpolation with piecewise different parameters called spline interpolation. As a matter of fact it is a kind of local interpolation since for each interval only neighboring points are considered.

In a spline of degree $n$ polynomial segments at most of degree $n$ are joined. The most simple case is when linear functions (first degree polynomials) are fitted between points. This is Euler's method of linear segments. Besides continuity at joints also differentiability of the function is frequently required. Such is a second degree spline, for example, when its derivatives at the joints computed from the left or right are the same. This is called a quadratic spline.

Generally a spline of degree $k$ and order $m$ is where piecewise polynomials of degree $k$ are fitted and these have order $m$ continuity at the joints. One of the most frequently used spline is cubic spline interpolation of second order. In this case third degree polynomials are fitted between data points that have common derivatives up to second order at the joints. There is a cubic spline interpolation of first order as well. In this case also third degree polynomials are fitted, but only the first derivatives are required to be the same (e.g. Hermite polynomial interpolation).

**LINEAR SPLINE INTERPOLATION**

Interpolation

Let us fit linear functions between data points. If \( n \) points are given there are \( n-1 \) intervals and so \( n-1 \) lines have to be determined. The most simple way is by using a Lagrange polynomial of two points:

\[
f_i(x) = \frac{x-x_{i+1}}{x_i-x_{i+1}} y_1 + \frac{x-x_i}{x_{i+1}-x_i} y_2
\]

Generally Matlab function \texttt{interp1} can be used for spline interpolation. It has a parameter for specifying which kind of spline interpolation is required. If not specified it defaults to linear interpolation. Possible interpolation methods for \texttt{interp1} are: 'linear' – default, 'nearest' – nearest neighbor, 'cubic' – cubic first order spline, 'spline' – cubic second order spline.

```
figure(2); plot(V,H,'r*'); hold on;
ylabel('water level [cm]');
sp = @(x) interp1(V,H,x);
g=ezplot(sp,[0,max(V)]);
xlabel('water volume [10^6 m^3]');
set(g,'Color','k')
```

Linear function fit gives a continuous interpolation, however its slope will have sudden changes. If smoother interpolation is required a higher order spline must be used.

---

**QUADRATIC SPLINE INTERPOLATION**

Quadratic (second degree) spline interpolation works by fitting quadratic polynomials \(( y = a·x^2 + b·x + c )\) into given data points and prescribing even continuity of first order derivatives computed from left or right.

A quadratic polynomial has 3 unknown coefficients which must be determined for \(( n-1 )\) intervals between \( n \) points. Therefore there are \( 3\cdot(n-1)=3n-3 \) unknowns to be determined from prescribed conditions, equations.

- Each polynomial must pass through the endpoints of its interval, hence there are two equations that can be specified as \( f_i(x) = a_i·x^2 + b_i·x + c_i \), altogether giving \( 2\cdot(n-1)=2n-2 \) equations.
- Derivatives taken at internal \(( n-2 )\) points \(( f'(x) = 2a·x + b \)\) are also the same: \(( n-2 )\) equations can be specified as \( 2a_{i-1}·x + b_{i-1} = 2a_i·x + b_i \).
- Because the above two conditions give only \(( 3n-4 )\) equations for \(( 3n-3 )\) unknowns, one additional condition must be given. We have several choices, for example set second derivative 0 at the first (or last) point: \( f_1''(x) = 2a_1 \), i.e. \( a_1 = 0 \). This effectively means that the first two (or the last two) points are connected by straight lines.

We must specify \(( 3n-4 )\) equations for quadratic spline fitting. Cubic second order splines are used more frequently, partly because also second derivatives

---

2 for homework

Interpolation

Numerical Methods, Lecture 10.

(curvatures) are continuous hence interpolation is also smoother, and partly because there is a formulation where only \((n-2)\) equations have to be solved. Matlab has no built-in functions for quadratic splines, only for cubics.

## CUBIC SECOND ORDER SPLINE INTERPOLATION

Consider now derivation of cubic, second order spline interpolation based on the formula of standard cubic polynomials. This is one of the most frequently used form of spline interpolation.

In this case third degree polynomials \(y=a \cdot x^3+b \cdot x^2+c \cdot x+d\) are fitted into the given points such that both first and second order derivatives are continuous at the joints. We have \((n-1)\) intervals in case of \(n\) points and for these intervals cubic polynomials have to be determined. Since each cubic polynomial has 4 unknown coefficients, there are \(4(n-1)=4n-4\) unknowns for which we must prescribe equations or conditions.

- Each polynomial must pass through the endpoints of its interval, hence there are two equations that can be specified as \(y_i=a_i \cdot x_i^3+b_i \cdot x_i^2+c_i \cdot x_i+d_i\), altogether giving \(2(n-1)=2n-2\) equations.
- Derivatives taken at internal \((n-2)\) points \((f'(x)=3ax^2+2bx+c)\) are also the same: \((n-2)\) equations can be specified as \(3a_{i-1} \cdot x_i^2+2b_{i-1} \cdot x_i+c_{i-1}=3a_i \cdot x_i^2+2b_i \cdot x_i+c_i\).
- Second derivatives taken at internal \((n-2)\) points \((f''(x)=6ax+2b)\) are also the same: additional \((n-2)\) equations can be specified as \(6a_{i-1} \cdot x_i+2b_{i-1}=6a_i \cdot x_i+2b_i\).
- Altogether we have thus \((4n-6)\) equations for \((4n-4)\) unknowns, therefore two additional conditions are to be specified. There are several choices. One possibility is to set second derivatives to 0 at the endpoints, this spline is called natural cubic spline. Another possibility that also used by Matlab is the "not-a-knot" condition. This means the continuity of third order derivatives at the second and next to the last points.

Remark: An alternative formulation of this cubic spline can also be derived. In that formulation the system of linear equations to be solved consists of only \((n-2)\) equations instead of \((4n-4)\). The derivation starts from second derivatives and the Lagrange form. See e.g.: Amos Gilat, Vish Subramaniam (2011): Numerical Methods, An Introduction with Applications Using MATLAB, John Wiley & Sons

Let us make cubic third degree spline interpolation of characteristic curve of the reservoir in Matlab. We will use \texttt{interp1} function again, but in this case we change the default 'linear' method to 'spline'.

```matlab
> sp2 = @(x) interp1(V,H,x,'spline')
> g=ezplot(sp2,[0,max(V)])
> set(g,'Color','r')
```
Interpolation

We see that our interpolation is smoother now. As it was mentioned, this interpolation is not by a natural cubic spline but by using the “not-a-knot” condition. This means the continuity of third order derivatives at the second and next to the last points. Since cubic polynomials are used, hence all parameters are identical within the first or last two intervals. This means that only one polynomial is fitted for the first or last 3 points. This is the reason behind the name “not-a-knot”: second and next to the last points are not true knots.

Most often this cubic, second order spline interpolation is used, hence there is a unique function called `spline` for the purpose. This function is equivalent to ‘spline’ method of `interp1`:

% What water height corresponds to 12 million m^3 volume?
H15 = sp2(12) % 2174 cm

% What volume belongs to 1500 cm water height?
f = @(x) sp2(x)-1500
v1500 = fzero(f,5) % 4.6699 million m^3

Matlab has another kind of cubic spline interpolation. The function `interp1` may as well be called with method ‘cubic’. This is cubic first order interpolation where still cubic polynomials are fitted inside each interval but only first order continuity is required. This is cubic Hermite interpolation and Matlab numerically differentiates the function to find derivatives at internal nodes by using 3 points or by using 2 points at the beginning/end of the function. (The same method like ‘cubic’ is ‘phchip’ – piecewise hermite interpolation.) Let us consider an example where it may be useful.
Interpolation

Numerical Methods, Lecture 10.

Our problem now relates to geodesy. To compute the Earth's gravity field we must know the Earth's internal density distribution. Earth's density ($\rho$) changes in terms of the radius ($R$) approximately like this:

<table>
<thead>
<tr>
<th>Radius [km]</th>
<th>0</th>
<th>800</th>
<th>1200</th>
<th>1400</th>
<th>2000</th>
<th>3000</th>
<th>3400</th>
<th>3600</th>
<th>4000</th>
<th>5000</th>
<th>5500</th>
<th>6370</th>
</tr>
</thead>
<tbody>
<tr>
<td>Density [kg/m$^3$]</td>
<td>13000</td>
<td>12900</td>
<td>12700</td>
<td>12000</td>
<td>11650</td>
<td>10600</td>
<td>9900</td>
<td>5500</td>
<td>5300</td>
<td>4750</td>
<td>4500</td>
<td>3300</td>
</tr>
</tbody>
</table>

Let us make a spline fit of the radius-density data and interpolate Earth's density at the radius of 3200 km, and radius where the density is exactly 4000 kg/m$^3$. First load earth_density.txt file and make cubic second order and cubic first order spline interpolation. Compare both interpolations in terms of best fit.

```matlab
> data = load('earth_density.txt')
> r = data(:,1) % radius in km
> ro = data(:,2) % density in kg/m^3

> % cubic first order Hermite interpolation
> fsp1 = @(x) interp1(r,ro,x,'cubic') % or 'pchip'
> fsp1(3200)
% 10361
> figure(1); subplot(1,2,1);
> plot(r,ro,'r*'); hold on;
> ezplot(fsp1,[0,max(r)]);
> title('Cubic first order Hermite interpolation')

> % cubic second order spline interpolation
> fsp2 = @(x) spline(r,ro,x)
> fsp2(3200)
% 11350
> subplot(1,2,2)
> plot(r,ro,'r*'); hold on;
> ezplot(fsp2,[0,max(r)]);
> title('Cubic second order spline interpolation')
```

The densities given by the two methods differ by much for radius of 3200 km, the first yielded 10361 kg/m$^3$, the second gave 11350 kg/m$^3$.

It is clear from the plots that cubic first order spline ('cubic') gave a much better fit than cubic second order spline ('spline'). It is because there are strong discontinuities, jumps in the data. When interpolating smooth functions, however, second order splines yield better results.
Interpolation Numerical Methods, Lecture 10.

Hence to answer the second question we should use our first spline to interpolate radius belonging to 4000 kg/m$^3$ density.

\[
\begin{align*}
\text{f} &= @(x) \ fsp1(x)-4000 \\
R4000 &= \text{fzero(f,5500)} \ % 5958.9 \ km
\end{align*}
\]

So approximately at 6000 km radius will density be 4000 kg/m$^3$. 