Opcion A

Jesus Molina Roldan

Exercici 1

Exercici I **Integració numèrica: Àrea dins una regió tancada**

Doneu una aproximació de l'àrea de la regió delimitada per la vostra ma.

Referència: http://www.mathworks.es/moler/chapters.html

1. Obteniu una imatge de la vostra ma. Seguiu les indicacions de l'exercici 3.4 de la pàgina 20 del capítol 3, "Interpolation"de Cleve Moler.

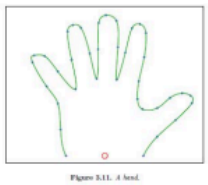


Figura 1:

2. Obteniu per interpolació $2D$ la corba que delimita la imatge de la vostra ma seguint els indicacions de l'exercici 3.4 i l'exercici 3.5 de les pàgines $20-21-22$. Responeu les preguntes que us formulen en els dos exercicis.

3. Què tan gran és la teva mà? Calcula l'àrea que ocupa la teva mà. Segueix les indicacions i respon les preguntes de l'exercici 6.23 de les pàgines $19-20$ del capítol 6 "Quadrature"de Cleve Moler.

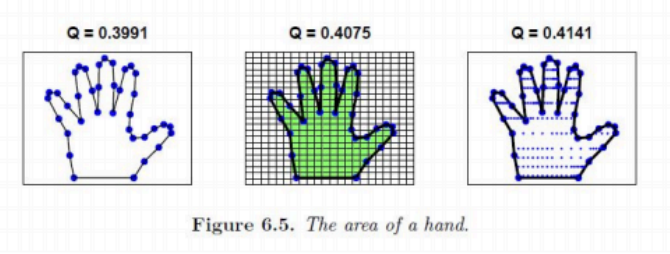


Figure 6.5. *The area of a hand.*

Figura 2: Moler - chpater 6 - exercici 6.23

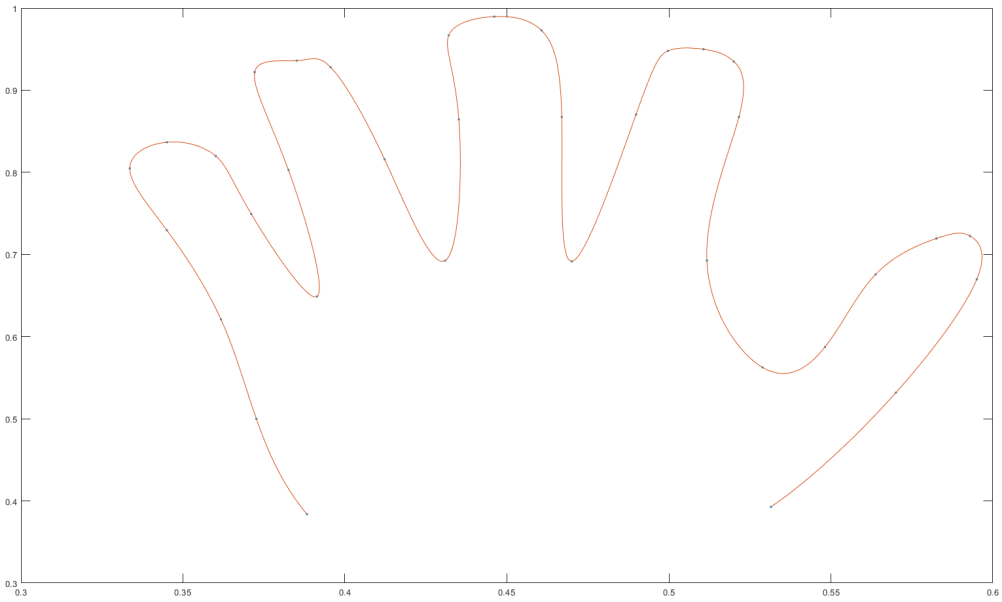
Exercici 1.2

```
clear
format long G
figure('position',get(0,'screensize'))
axes('position',[0 0 1 1])
%[x,y] = ginput;
load myhand.mat
```

Splinetx

```
n = length(x);
s = (1:n)';
t = (1:.05:n)';
u = splinetx(s,x,t);
v = splinetx(s,y,t);
clf reset
```

```
plot(x,y,'.',u,v,'-');
```



pchiptx

```
n = length(x);
s = (1:n)';
t = (1:.05:n)';
u = pchiptx(s,x,t);
v = pchiptx(s,y,t);
clf reset
plot(x,y,'.',u,v,'-');
```

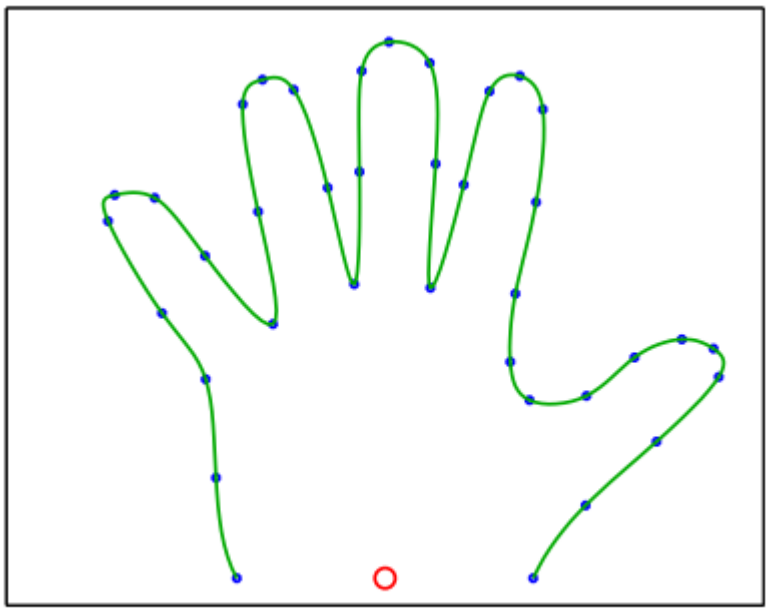


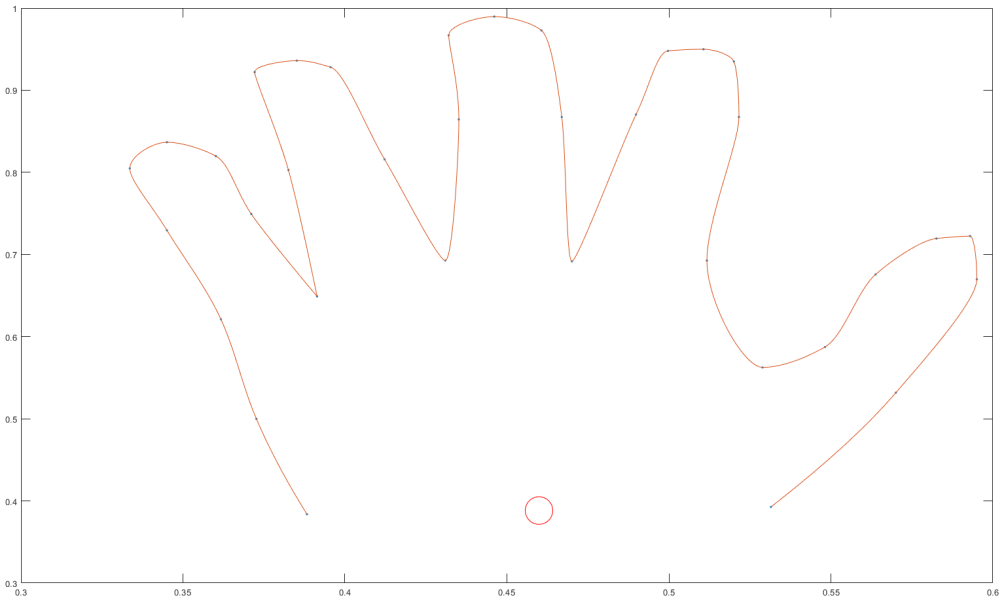
Figure 3.11. *A hand.*

Do the same thing with `pchiptx`. Which do you prefer?
Figure 3.11 is the plot of my hand. Can you tell if it was done with `splinetx` or `pchiptx`?

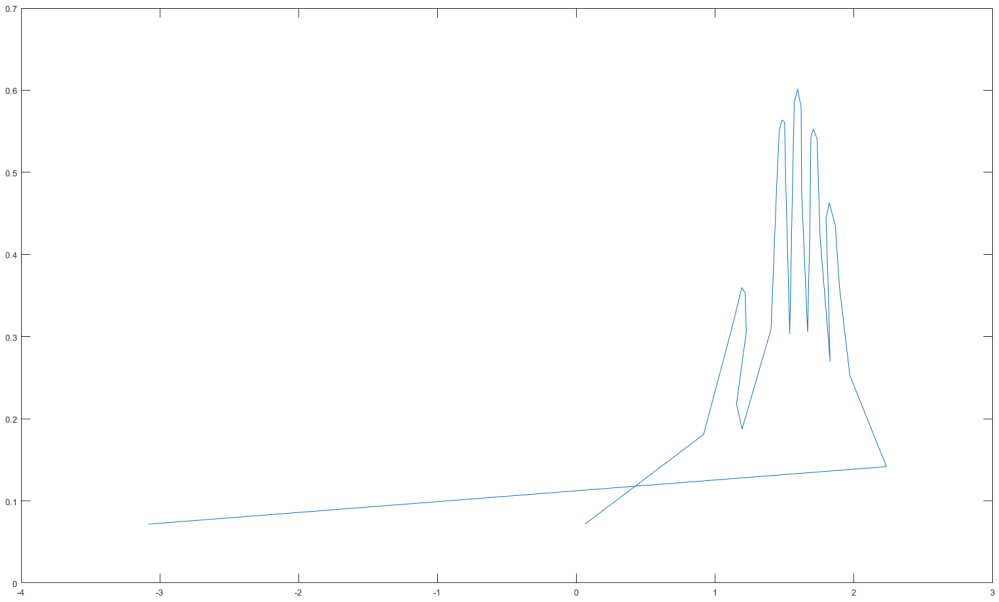
Mas o menos los plots de las manos son parecidas. Aun así prefiero la interpolacion con splinetx ya que hace mejor la forma curvada de la punta de los dedos. Creo que la figura 3.11 se ha hecho con splinetx.

3.5. The previous exercise uses the data index number as the independent variable for two-dimensional parametric interpolation. This exercise uses, instead, the angle θ from polar coordinates. In order to do this, the data must be centered so that they lie on a curve that is *starlike* with respect to the origin, that is, every ray emanating from the origin meets the data only once. This means that you must be able to find values x_0 and y_0 so that the MATLAB statements

```
x0 = ((x(end)-x(1))/2)+x(1);
y0 = ((y(end)-y(1))/2)+y(1);
hold on
scatter(x0,y0,1000, 'red');
hold off
```




```
x2 = x -x0;
y2 = y - y0;
theta = atan2(y2,x2);
r = sqrt(x2.^2 + y2.^2);
plot(theta,r);
```



produce a set of points that can be interpolated with a single-valued function, $r = r(\theta)$. For the data obtained by sampling the outline of your hand, the point (x_0, y_0) is located near the base of your palm. See the small circle in Figure 3.11. Furthermore, in order to use **splinetx** and **pchiptx**, it is also necessary to order the data so that **theta** is monotonically increasing. Choose a subsampling increment, **delta**, and let

```
delta = .05;
t = (theta(1):delta:theta(end))';
p = pchiptx(theta,r,t);
s = splinetx(theta,r,t);
```

Examine two plots:

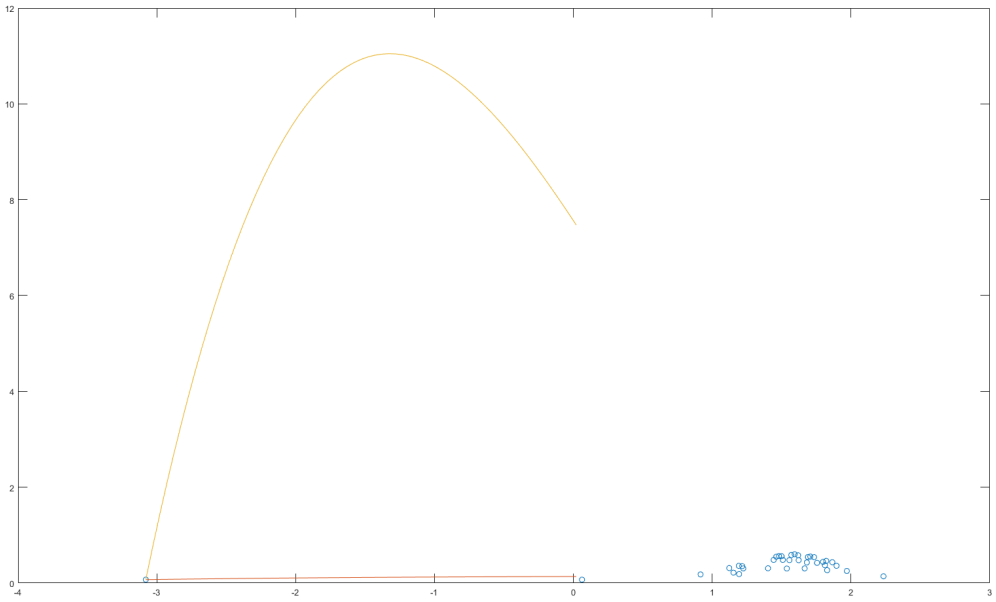
```
plot(theta,r,'o',t,[p s],'-')
```

and

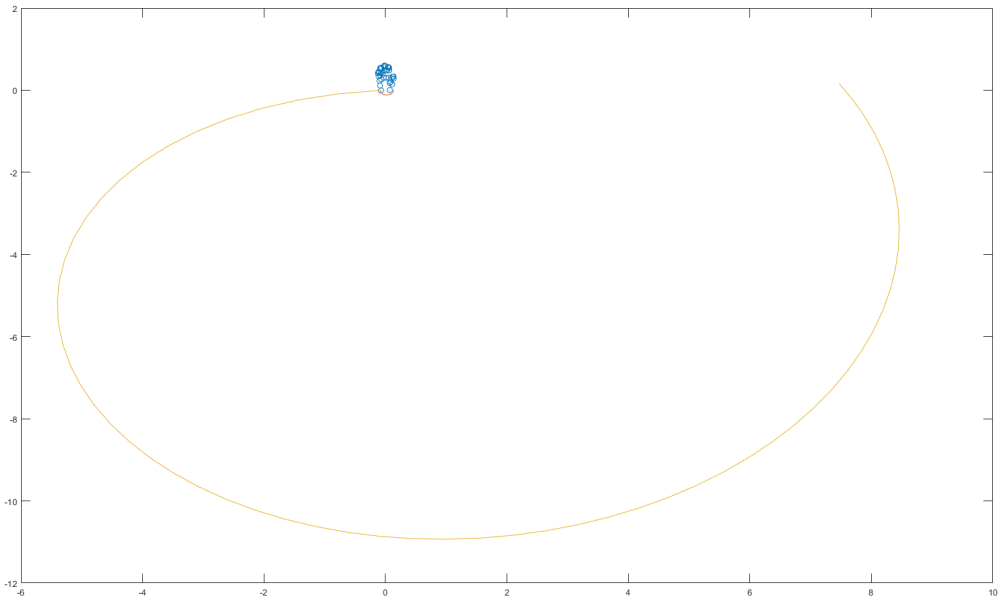
```
plot(x,y,'o',p.*cos(t),p.*sin(t),'-',...
       s.*cos(t),s.*sin(t),'-')
```

Compare this approach with the one used in the previous exercise. Which do you prefer? Why?

```
plot(theta,r,'o',t,[p s],'-')
```




```
plot(x2,y2,'o',p.*cos(t),p.*sin(t),'-',...
s.*cos(t),s.*sin(t),'-')
```



Exercici 1.2

6.23. How large is your hand? Figure 6.5 shows three different approaches to computing the area enclosed by the data that you obtained for exercise 3.3.

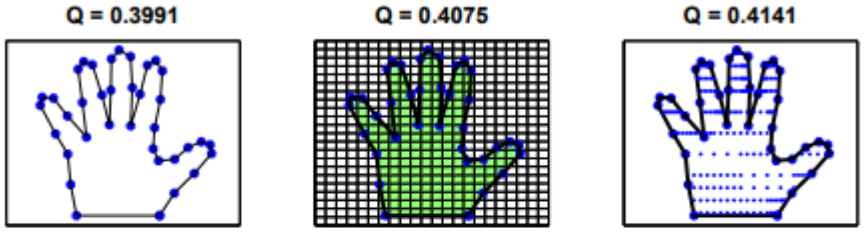


Figure 6.5. *The area of a hand.*

(a) Area of a polygon. Connect successive data points with straight lines and connect the last data point to the first. If none of these lines intersect, the result is a polygon with n vertices, (x_i, y_i). A classic, but little known, fact is that the area of this polygon is

$$(x_1y_2 - x_2y_1 + x_2y_3 - x_3y_2 + \cdots + x_ny_1 - x_1y_n)/2.$$

If x and y are column vectors, this can be computed with the MATLAB one-liner

```
area = (x'*y([2:n 1]) - x([2:n 1])'*y)/2
```

```
area =
       -0.0859964622641511
```

(b) Simple quadrature. The MATLAB function `inpolygon` determines which of a set of points is contained in a given polygonal region in the plane. The polygon is specified by the two arrays `x` and `y` containing the coordinates of the vertices. The set of points can be a two-dimensional square grid with spacing `h`.

```
[u,v] = meshgrid(xmin:h:xmax,ymin:h:ymax)
```

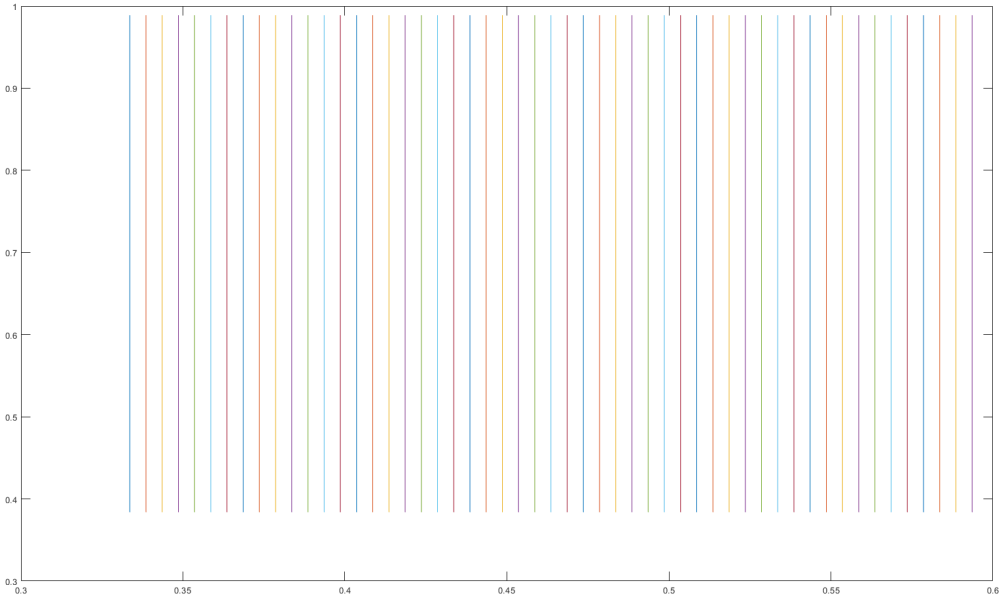
The statement

```
k = inpolygon(u,v,x,y)
```

returns an array the same size as `u` and `v` whose elements are one for the points in the polygon and zero for the points outside. The total number of points in the region is the number of nonzeros in `k`, that is, `nnz(k)`, so the area of the corresponding portion of the grid is

```
h^2*nnz(k)
```

```
h = 0.005;
xmin = min(x);
xmax = max(x);
ymin = min(y);
ymax = max(y);
[u, v] = meshgrid(xmin:h:xmax,ymin:h:ymax);
k = inpolygon(u,v,x,y);
plot(u,v);
```




```
nnz(k);
radi = (h^2)*nnz(k)
```

```
radi =
                0.0859
```

(c) Two-dimensional adaptive quadrature. The *characteristic function* of the region $\chi(u,v)$ is equal to one for points (u,v) in the region and zero for points outside. The area of the region is

$$\int\int \chi(u,v)dudv.$$

The MATLAB function `inpolygon(u,v,x,y)` computes the characteristic function if `u` and `v` are scalars, or arrays of the same size. But the quadrature functions have one of them a scalar and the other an array. So we need an M-file, `chi.m`, containing

```
function k = chi(u,v,x,y)
if all(size(u) == 1), u = u(ones(size(v))); end
if all(size(v) == 1), v = v(ones(size(u))); end
k = inpolygon(u,v,x,y);
```

Two-dimensional adaptive numerical quadrature is obtained with

```
x = ..;
y = ..;
dblquad(@(u,v)chi(u,v,x,y),xmin,xmax,ymin,ymax,tol)
function k = chi(u,v,x,y)
```

This is the least efficient of the three methods. Adaptive quadrature expects the integrand to be reasonably smooth, but $\chi(u,v)$ is certainly not smooth. Consequently, values of `tol` smaller than 10^{-4} or 10^{-5} require a lot of computer time.
Figure 6.5 shows that the estimates of the area obtained by these three methods agree to about two digits, even with fairly large grid sizes and tolerances. Experiment with your own data, use a moderate amount of computer time, and see how close the three estimates can be to each other.

```
tol = 10^(-4);
radi = dblquad(@(u,v)chi(u,v,x,y),xmin,xmax,ymin,ymax,tol)
```

```
radi =
        0.0863943030071044
```

Exercici 2

Exercici II **Opció A**

Les dades de la taula següent estan relacionats amb l'esperança de vida al nèixer dels ciutadants de *Grècia*:

any	1975	1980	1985	1990	1995	2000	2005	2010
Grècia	72.3	73.6	75.1	77.0	77.6	77.9	79.2	80.4

Es demana:

Numèricament és millor que considereu la taula inicial amb abscisses $0, 1, \cdots, 7.$, o que centreu les dades. Altrement els resultats no són correctes!!

(2.a) Useu el polinomi interpolador de grau 7 (escalat) per estimar l'esperança de vida el 1970, 1992, 2007. Compareu els valors obtinguts, amb les xifres oficials.

(2.b) Useu un *spline natural* per estimar l'esperança de vida el 1970, 1992, 2007. Compareu els valors obtinguts, amb les xifres oficials.

(2.c) Busqueu un polinomi de grau més petit que 5 per mínims quadrats. Justifiqueu l'elecció mostrant una cota de l'error d'aquest i de la resta amb els que heu provat. Compareu els valors obtinguts, amb les xifres oficials per cada pais.

(2.d) Extrapoleu un valor per l'any 1970 i un per l'any 2015 pels tres models obtinguts. Recordant les dades oficials, i els resultats obtinguts els models estudiats són vàlids per estimar amb precissió l'esperança de vida per l'any 1970? I per l'any 2015?

(2.e) Feu una gràfica on apareguin les dades (representats per una rodona) i les totes solucions trobades.

Taula amb les dades per comparar

any	1970	1992	2007	2015
Grècia	70.9	77.4	79.4	81.6

2.a

```
clear
format long G
x = [1975 1980 1985 1990 1995 2000 2005 2010];
y = [72.3 73.6 75.1 77.0 77.6 77.9 79.2 80.4];
anys = [1970 1992 2007 2015];
valor_exacto = [70.9 77.4 49.4 81.6];
```

Polinomi interpolador grau 7

```
grau = 7;
[p, ~, mu] = polyfit(x,y,grau);
aproximado = polyval(p,anys,[], mu);
error = error_absoluto(valor_exacto,aproximado);
t = array2table(p, 'RowNames', {'coeficient polinomi'})
```

```
t = 1×8 table
```

...

	p1	p2	p3	p4	p5	p6	p7
1 coeficient polinomi	0.70335...	-1.0950...	-2.7893...	3.84687...	3.50060...	-3.8298...	1.32862...

```
t1 = array2table([valor_exacto;aproximado; error], 'VariableNames', {'1970' ,'1992' ,'2007' ,'2
```

t1 = 3×4 table

	1970	1992	2007	2015
1 valor	70.9	77.4	49.4	81.6
2 aproximat	50.7999...	77.3921...	79.8143...	81.7000...
3 desviat	20.1000...	0.00787...	30.4143...	0.10000...

2.b

Spline

```
x = (x-1975)/(35/7);
anys2 = (anys-1975)/(35/7);
aproximado = interp1(x,y,anys2,'spline');
error = error_absoluto(valor_exacto,aproximado);
t2 = array2table([valor_exacto;aproximado; error], 'VariableNames', {'1970' ,'1992' ,'2007' ,'2
```

t2 = 3×4 table

	1970	1992	2007	2015
1 valor	70.9	77.4	49.4	81.6
2 aproximat	70.3143...	77.3947...	79.7874...	79.7961...
3 desviat	0.58564...	0.00521...	30.3874...	1.80382...

2.c

Polinomio minimos cuadrados

```
n = 5;
error = zeros(n,length(anys2)+1);
RowNames = {};
for i = 1:n
    coeficientes = polinomio_interpolador_minimos_cuadrados(x, y, i);
    aproximado = polyval(coeficientes,anys2);
    RowNames{i} = strcat('grau ', int2str(i));
    e = error_absoluto(valor_exacto, aproximado);
    error(i,:) = [e, sum(e,2)];
end
t3 = array2table(error,'VariableNames', {'1970' ,'1992' ,'2007' ,'2015', 'Desviacio'}, 'RowName
```

t3 = 5×5 table

	1970	1992	2007	2015	Desviacio
1 grau1	0.71785...	0.87404...	30.4723...	0.05714...	32.12142857...
2 grau2	0.20178...	0.55278...	30.2786...	0.86249...	31.89571428...

	1970	1992	2007	2015	Desviacio
3 grau3	1.11428...	0.53575...	30.2337...	0.04999...	31.93377489...
4 grau4	1.28035...	0.27564...	29.9764...	2.44464...	33.97705281...
5 grau5	4.32499...	0.25249...	30.2022...	0.59999...	35.37976783...

Como podemos ver el polinomio de minimos cuadrados que da mejor resultado es el de grado 2, ya que es el que tiene menor error absoluto.

2.d

Como podemos ver en los resultados del polinomio de grado 7 se desvia mucho del valor correcto respecto al año 1970. Por lo que consideraria que no es muy valido. En contraposición el polinomio realizado con spline es mejor ya que la desviación del año 1970 es de 0.6 y la del 2015 de 1.8. El polinomio de grado 3 realizado con minimos cuadrados es el que da mejor resultado ya que el error absoluto del año 1970 es de 1.11 y el del año 2015 ,0.049, dando como cifras correctas 1.

2.e

Dades

```
x = [1975 1980 1985 1990 1995 2000 2005 2010];
y = [72.3 73.6 75.1 77.0 77.6 77.9 79.2 80.4];
scatter(x,y, 'DisplayName','Dades 1');
anys = [1970 1992 2007 2015];
valor_exacto = [70.9 77.4 49.4 81.6];
hold on;
scatter(anys, valor_exacto, 'DisplayName','Dades 2');
```

Polinomi interpolador grau 7(PLOT)

```
grau = 7;
[coeficientes, ~, mu] = polyfit(x,y,grau);
xplot = 1970:2015;
yplot = polyval(coeficientes,xplot,[], mu);
hold on;
plot(xplot,yplot, 'DisplayName','polinomi grau 7');
```

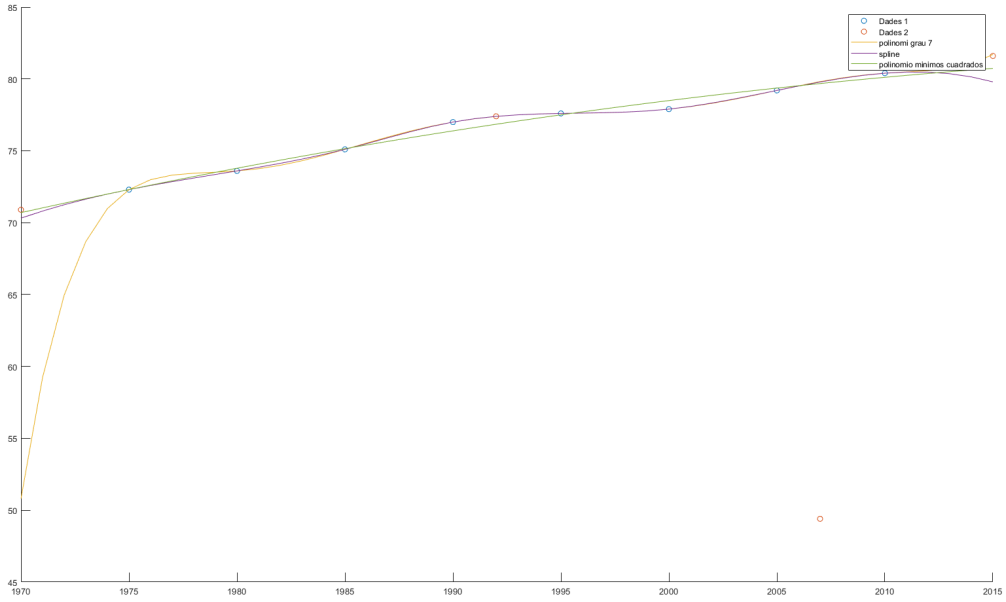
Spline(PLOT)

```
x = (x-1975)/(35/7);
xplot_centrado = ((1970:2015)-1975)/(35/7);
yplot = interp1(x,y,xplot_centrado,'spline');
xplot = 1970:2015;
hold on;
plot(xplot,yplot, 'DisplayName','spline');
```

Polinomio minimos cuadrados(Plot)

```
grado = 2;
[coeficientes] = polinomio_interpolador_minimos_cuadrados(x, y, grado);
```

```
xplot_centrado = ((1970:2015)-1975)/(35/7);
yplot = polyval(coeficientes,xplot_centrado);
xplot = 1970:2015;
hold on;
plot(xplot,yplot, 'DisplayName','polinomio minimos cuadrados');
legend
hold off;
```



Exercici 3

Exercici III **Opció A**

La derivada de la funció $f(x) = \arctan\left(\frac{x}{5}\right)$ en $x = \sqrt{5}$ pren el valor $f'(\sqrt{5}) = 1/6$.

(3.a) Aproximeu $f'(\sqrt{5})$ fent ús de la fórmula d'ordre $\mathcal{O}(h^2)$

$$\frac{f(x_0+h) - f(x_0-h)}{2h},$$

per a $h_k = 2^{-k}$ i $k = 1, 2 \cdots 5$. Calculeu l'error absolut per cada una de les aproximacions obtingudes.

(3.b) Feu ús de la tècnica d'Extrapolació de Richardson d'ordre $\mathcal{O}(h^2)$ per obtenir una aproximació de $f'(\sqrt{5})$ amb més decimals correctes Calculeu l'error absolut per cada una de les aproximacions obtingudes.

(3.c) Presenteu els resultats dels dos apartats previs en una mateixa taula (T1).

(3.d) Representeu les xifres decimals correctes en una gràfica, amb $k = 1, 2 \ldots 5$ a l'eix d'abscises. Quantes xifres decimals correctes s'obtenen en cada cas? Argumenta la teva resposta.

3.a

Central difference approximation

```
clear
format long G
syms f(x) x
f(x) = atan(x/5);
point = sqrt(5);
iterations = 5;
df = diff(f,x);
dfx = double(df(point));
N = 1;
[T, resultat] = extrapolation_richardson(f,point,iterations, N);
t = array2table(T,'VariableNames',{'baske^k', 'N1','error'})
```

t = 5×3 table

	baske^k	N1	error
1	0.5	0.166511...	0.000155...
2	0.25	0.166628...	3.866039149...
3	0.125	0.166657...	9.650081586...
4	0.0625	0.166664...	2.411579342...
5	0.03125	0.166666...	6.028359800...

Extrapolation richardson

3.b

```
N = 5;
[T, resultat] = extrapolation_richardson(f,point,iterations, N);
t = array2table(T,'VariableNames', {'base^k', 'N1', 'N2', 'N3', 'N4', 'N5', 'error'})
```

t = 5×7 table

	base^k	N1	N2	N3	N4	N5	error
1	0.5	0.166511...	0	0	0	0	0
2	0.25	0.166628...	0.166666...	0	0	0	0
3	0.125	0.166657...	0.166666...	0.166666...	0	0	0
4	0.0625	0.166664...	0.166666...	0.166666...	0.166666...	0	0
5	0.03125	0.166666...	0.166666...	0.166666...	0.166666...	0.166666...	2.775557561...

3.c

N1 consiste en la central difference approximation por lo tanto ya he mostrado una tabla con los dos apartados anteriores en el ejercicio 3b.

3.d

```
iterations = 5;
point = sqrt(5);
```

```
N_total = 5;
w = 1;
for N = 1:N_total
    [T, resultat] = extrapolation_richardson(f,point,iterations, N);
    VariableNames = {'base^k'};
    for i = 1:N
        VariableNames{i+1} = strcat('N ', int2str(i));
    end
    VariableNames{end+1} = 'error';
    t = array2table(T,'VariableNames',VariableNames)
    xplot = w:N_total;
    yplot = xifres_decimals_correctes(1/6, resultat(w:end,w));
    plot(xplot,yplot, 'DisplayName', strcat('N',int2str(N),'(h)'));
    if(N == 1)
        hold on;
    end
    w = w + 1;
end
```

t = 5×3 table

	base^k	N1	error
1	0.5	0.166511...	0.000155...
2	0.25	0.166628...	3.866039149...
3	0.125	0.166657...	9.650081586...
4	0.0625	0.166664...	2.411579342...
5	0.03125	0.166666...	6.028359800...

t = 5×4 table

	base^k	N1	N2	error
1	0.5	0.166511...	0	0
2	0.25	0.166628...	0.166666...	3.169029416...
3	0.125	0.166657...	0.166666...	2.002171670...
4	0.0625	0.166664...	0.166666...	1.254738701...
5	0.03125	0.166666...	0.166666...	7.847408833...

t = 5×5 table

	base^k	N1	N2	N3	error
1	0.5	0.166511...	0	0	0
2	0.25	0.166628...	0.166666...	0	0
3	0.125	0.166657...	0.166666...	0.166666...	2.296350332...
4	0.0625	0.166664...	0.166666...	0.166666...	3.606837051...
5	0.03125	0.166666...	0.166666...	0.166666...	5.642708522...

t = 5×6 table

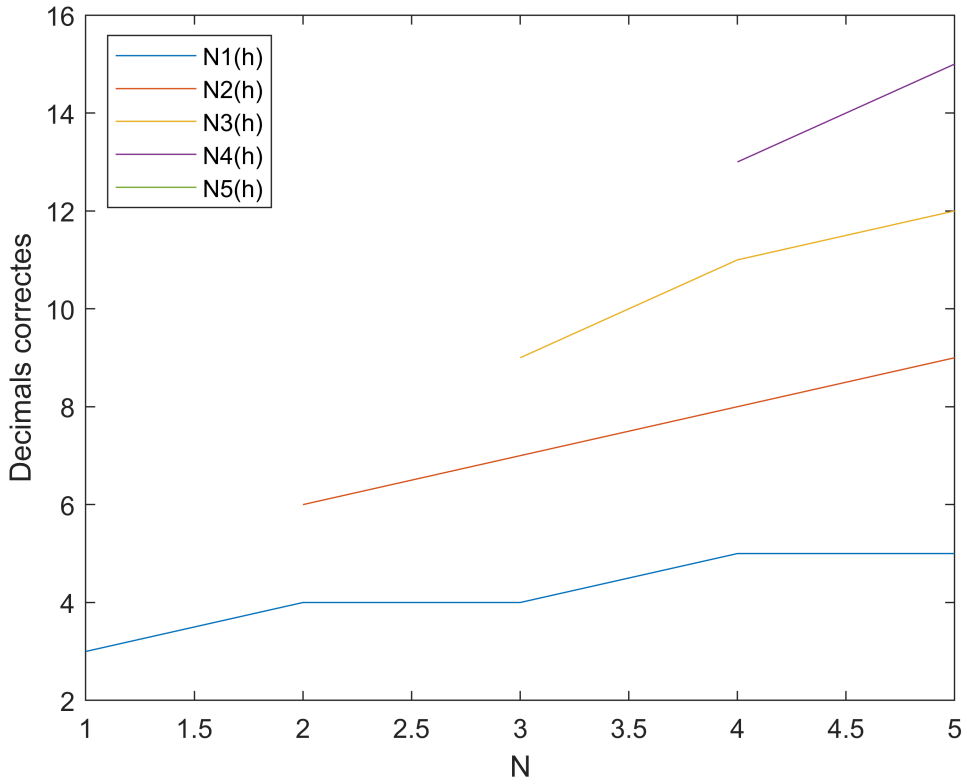
	base^k	N1	N2	N3	N4	error
1	0.5	0.166511...	0	0	0	0
2	0.25	0.166628...	0.166666...	0	0	0

	base^k	N1	N2	N3	N4	error
3	0.125	0.166657...	0.166666...	0.166666...	0	0
4	0.0625	0.166664...	0.166666...	0.166666...	0.166666...	1.909583602...
5	0.03125	0.166666...	0.166666...	0.166666...	0.166666...	5.551115123...

t = 5×7 table

	base^k	N1	N2	N3	N4	N5	error
1	0.5	0.166511...	0	0	0	0	0
2	0.25	0.166628...	0.166666...	0	0	0	0
3	0.125	0.166657...	0.166666...	0.166666...	0	0	0
4	0.0625	0.166664...	0.166666...	0.166666...	0.166666...	0	0
5	0.03125	0.166666...	0.166666...	0.166666...	0.166666...	0.166666...	2.775557561...

```
legend('Location', 'northwest');
ylabel('Decimals correctes'), xlabel('N')
hold off
```



Exercici IV **Opció A**

El logaritme neperià potset calculat fent ús de la fórmula

$$\ln(x-1) = \int_2^x \frac{1}{x-1}\,dx\,, \quad x > 2\,. \tag{IV-1}$$

(**4.a**) Useu la regla composta de Trapezis per a determinar $\ln(2)$ a partir de l'expressió (IV-1) i $h = 2^{-k}$, $k = 0, 1, 2, 3, 4, 5$. Presenteu els resultats en taules.

(**4.b**) Apliqueu el mètode de Romberg a les aproximacions per trapezis de l'apartat (**4.a**) per millorar l'aproximació de $\ln(2)$ obtinguda. Presenteu els resultats en taules.

(**4.c**) Useu una tècnica de simulació (Mètode de MonteCarlo) per a determinar $\ln(2)$ preneu mostres de mida 10^{-k}, $k \geq 3, 4, 5, 6, \ldots$. Presenteu els resultats que s'obtenen taules.

(**4.d**) Useu $\ln(x)$ de MATLAB® per obtenir el valor de $\ln(2)$ amb 15 xifres decimals correctes, calculeu l'error absolut i l'error r elatiu de les aproximació dels apartats (**4.a**), (**4.b**) i (**4.c**). Quantes xifres significatives correctes s'han obtingut? Explica els teus càlculs.

(**4.e**) Com s'ha de pendre la mostra de gran per obtenir la mateixa exactitud que amb la fórmula dels trapezis? Podem obtenir la mateixa exactitud que amb la fórmula de Simpson (segona columna de Romberg)? Argumenta la teva resposta

4.a

Regla composta de Trapezis

```
clear
format long G
f = @(x) 1./(x-1);
base = 1/2;
iterations = 6;
a = 2;
b = 3;
[T, ~] = regla_composta_trapezis_iterations(f, a, b, base, iterations);
t = array2table(T,'VariableNames', {'k' ,'h' ,'valor' ,'error'})
```

t = 6×4 table

	k	h	valor	error
1	0	1	0.75	0.05685...
2	1	0.5	0.70833...	0.01518...
3	2	0.25	0.69702...	0.00387...
4	3	0.125	0.69412...	0.00097...
5	4	0.0625	0.69339...	0.00024...
6	5	0.03125	0.69320...	6.10277093...

4.b

Metode Romberg

```
N = 4;
[T , ~] = metode_romberg(f,a,b,iterations, N);
VariableNames = {'k', 'h'};
for i = 1:N
    VariableNames{end+1} = strcat('R_', int2str(i));
end
VariableNames{end+1} = 'errAbs';
t = array2table(T, 'VariableNames', VariableNames)
```

t = 6×7 table

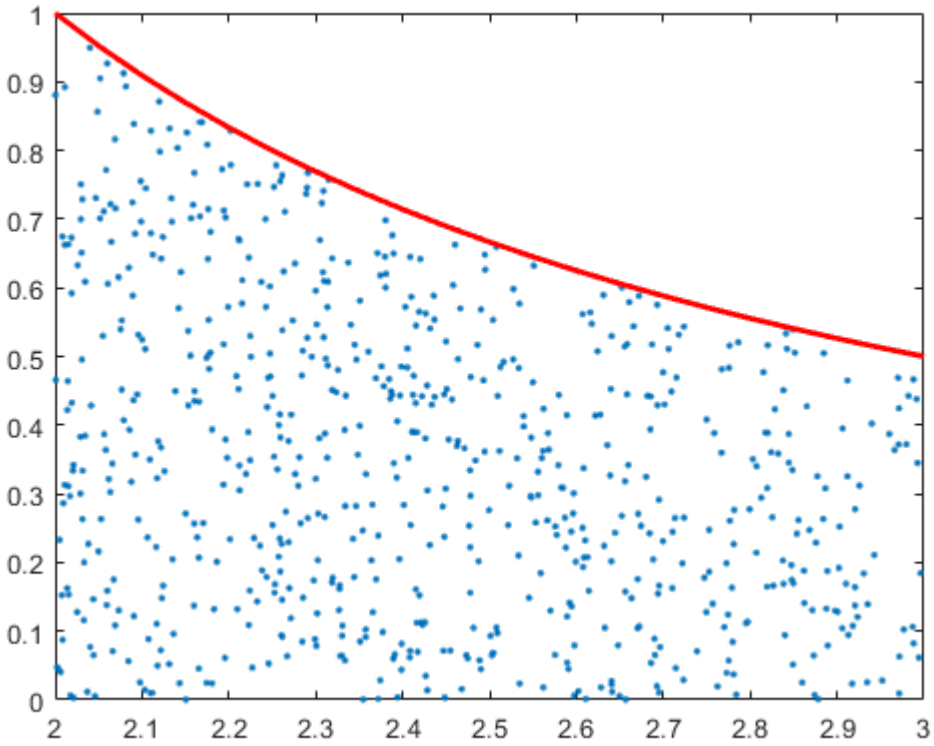
	k	h	R_1	R_2	R_3	R_4	errAbs
1	0	0.5	0.75	0	0	0	0
2	1	0.25	0.70833...	0.69444...	0	0	0
3	2	0.125	0.69702...	0.69325...	0.69317...	0	0
4	3	0.0625	0.69412...	0.69315...	0.69314...	0.69314...	2.970848866...
5	4	0.03125	0.69339...	0.69314...	0.69314...	0.69314...	2.511987284...
6	5	0.015625	0.69320...	0.69314...	0.69314...	0.69314...	1.347200129...

4.c

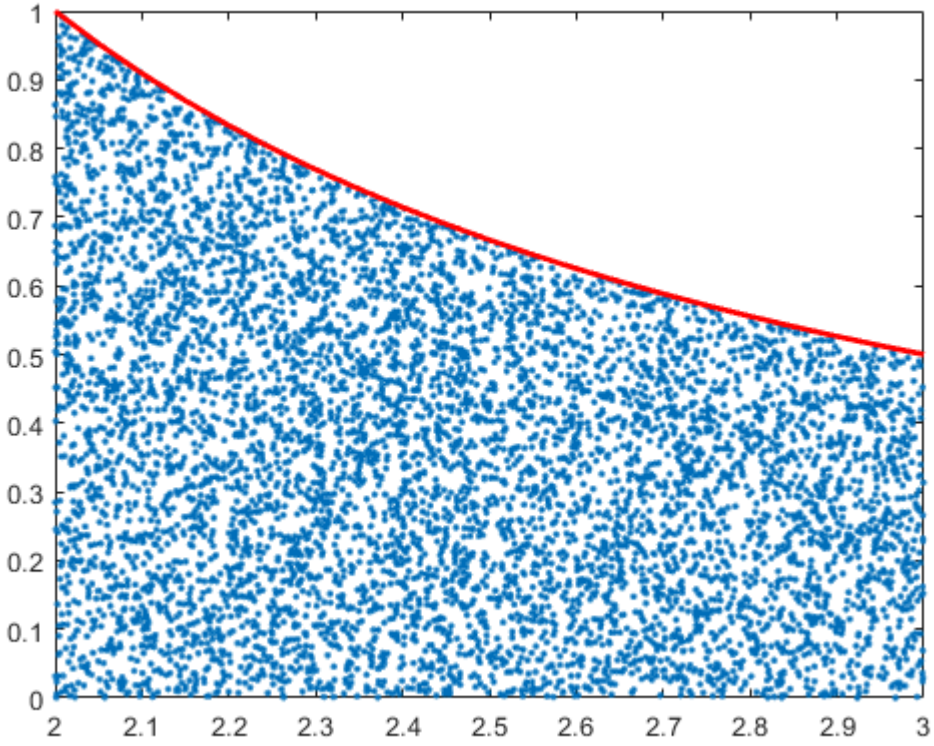
Metode Montecarlo

```
T = zeros(5,1);
for k = 3:7
    M = 10^k
    [resultat, points] = metode_montecarlo(f,a,b, M);
    T(k-2) = resultat;
    t = a:0.05:b; yt = f(t);
    figure
    plot(points(:,1),points(:,2),'.',t,yt,'r','LineWidth',2)
end
```

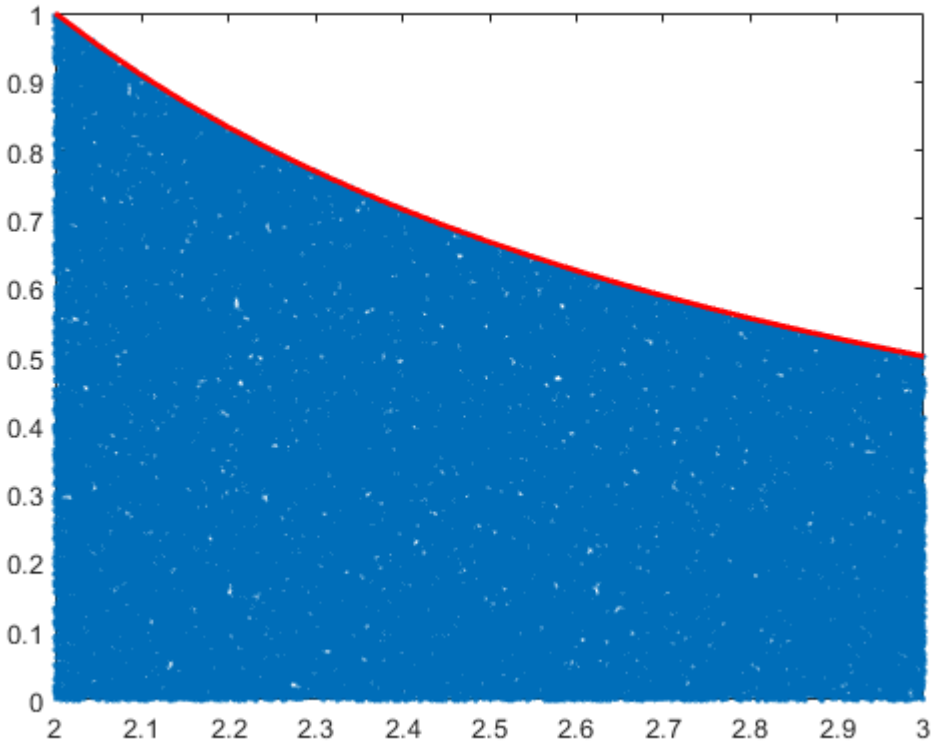
```
M =
        1000
```



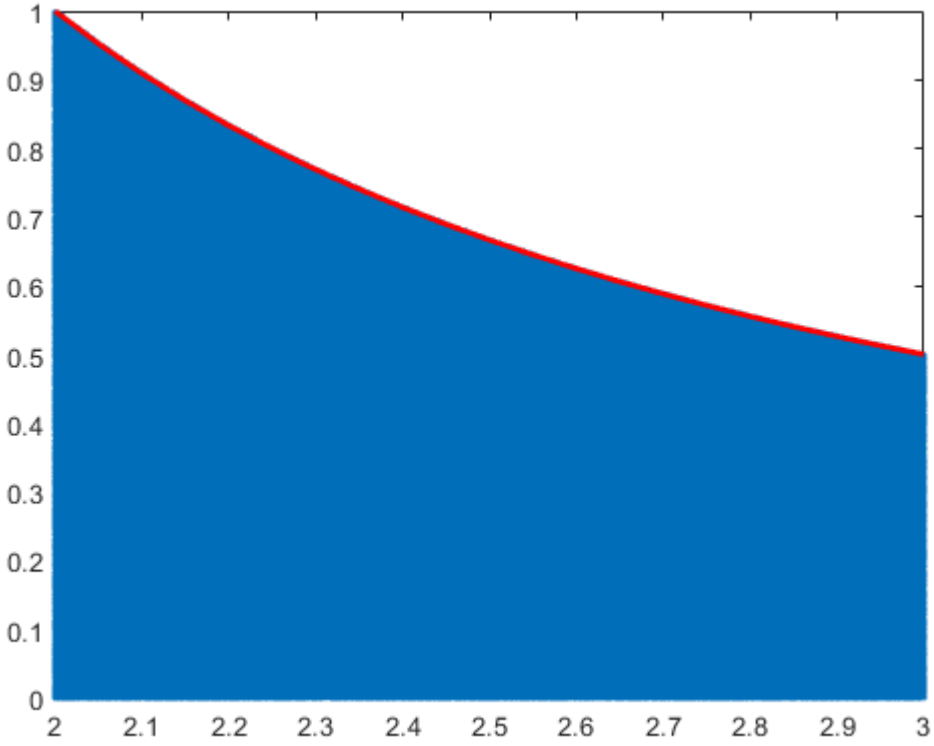

```
M =
        10000
```



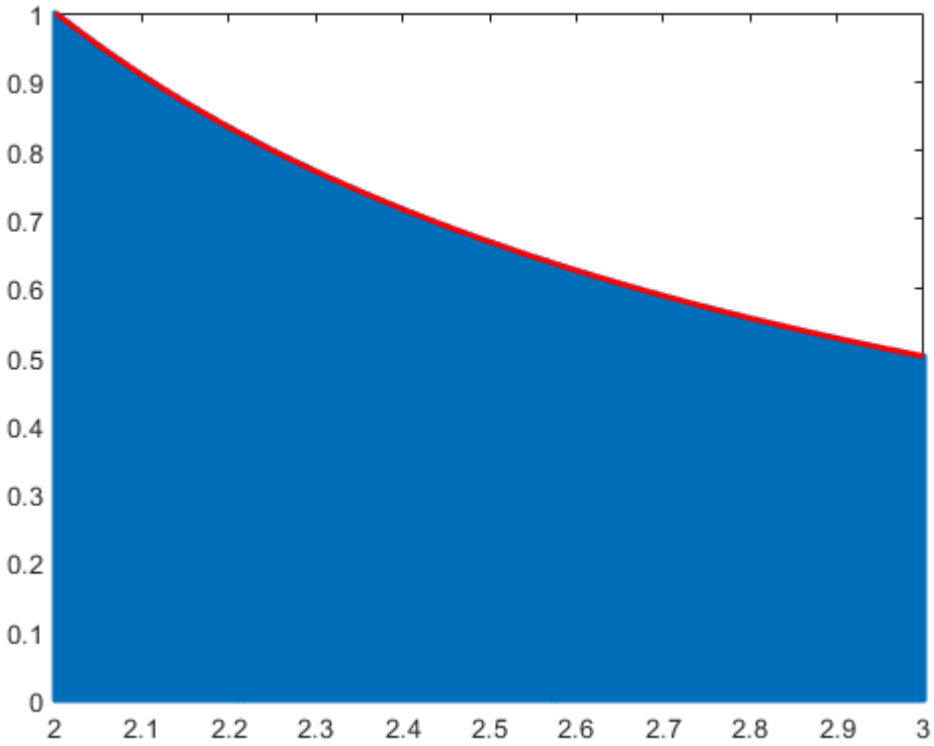

```
M =
        100000
```




```
M =
     1000000
```




```
M =
    10000000
```




```
k = (3:7)';
M = (ones(5,1)*10).^k;
T = [k M T error_absoluto(T,log(2))];
t = array2table(T, 'VariableNames', {'k', 'M', 'va_ln2', 'errAbs'})
```

t = 5×4 table

	k	M	va_ln2	errAbs
1	3	1000	0.696	0.002852...
2	4	10000	0.6891	0.004047...
3	5	100000	0.69309	5.718055994...
4	6	1000000	0.69347	0.000322...
5	7	10000000	0.6933562	0.000209...

4.d

Errors

Error Trapezis

```
k = (0:5)';
[~, resultat] = regla_composta_trapezis_iterations(f, a, b, base, iterations);
error_absoluto_trapezis = error_absoluto(log(2),resultat);
error_relativo_trapezis = error_relativo(log(2),resultat);
xifres = xifres_significatives_correctes(log(2),resultat);
T = [k error_absoluto_trapezis error_relativo_trapezis xifres];
```

```
t = array2table(T, 'VariableNames',{'k', 'errAbs', 'errRel', 'xifres'})
```

t = 6×4 table

	k	errAbs	errRel	xifres
1	0	0.056852...	0.082021...	0
2	1	0.015186...	0.021908...	1
3	2	0.003876...	0.005592...	1
4	3	0.000974...	0.001406...	2
5	4	0.000244...	0.000352...	3
6	5	6.102770930...	8.804437356...	3

Error absolut Romberg

```
N = 4;
h = (1/2).^k;
[~ ,taula_romberg] = metode_romberg(f,a,b, iterations, N);
error_absoluto_romberg = error_absoluto(log(2),taula_romberg);
for i = 1:iterations
    for j = (i+1):N
        error_absoluto_romberg(i,j) = 0;
    end
end
VariableNames = {'k', 'h'};
for i = 1:N
    VariableNames{end+1} = strcat('ER_', int2str(i));
end
T = [k h error_absoluto_romberg];
t = array2table(T, 'VariableNames',VariableNames)
```

t = 6×6 table

	k	h	ER_1	ER_2	ER_3	ER_4
1	0	1	0.05685...	0	0	0
2	1	0.5	0.01518...	0.00129...	0	0
3	2	0.25	0.00387...	0.00010...	2.74226146...	0
4	3	0.125	0.00097...	7.35009458...	7.20921289...	2.97084886...
5	4	0.0625	0.00024...	4.72259473...	1.37371327...	2.51198750...
6	5	0.03125	6.10277093...	2.97298774...	2.27904362...	1.34722233...

Error relatiu Romberg

```
error_relativo_romberg = error_relativo(log(2),taula_romberg);
for i = 1:iterations
    for j = (i+1):N
        error_relativo_romberg(i,j) = 0;
    end
end
VariableNames = {'k', 'h'};
```

```
for i = 1:N
    VariableNames{end+1} = strcat('ER_', int2str(i));
end
T = [k h error_relativo_romberg];
t = array2table(T, 'VariableNames',VariableNames)
```

t = 6×6 table

	k	h	ER_1	ER_2	ER_3	ER_4
1	0	1	0.08202...	0	0	0
2	1	0.5	0.02190...	0.00187...	0	0
3	2	0.25	0.00559...	0.00015...	3.95624701...	0
4	3	0.125	0.00140...	1.06039450...	1.04006956...	4.28602893...
5	4	0.0625	0.00035...	6.81326400...	1.98184933...	3.62403191...
6	5	0.03125	8.80443735...	4.28911467...	3.28796492...	1.94363097...

Xifres significatives correctes Romberg

```
xifressignificativescorrectes = xifres_significatives_correctes(log(2),taula_romberg);
for i = 1:iterations
    for j = (i+1):N
        xifressignificativescorrectes(i,j) = 0;
    end
end
VariableNames = {'k', 'h'};
for i = 1:N
    VariableNames{end+1} = strcat('xifres R_', int2str(i));
end
T = [k h xifressignificativescorrectes];
t = array2table(T, 'VariableNames',VariableNames)
```

t = 6×6 table

	k	h	xifres R_1	xifres R_2	xifres R_3	xifres R_4
1	0	1	0	0	0	0
2	1	0.5	1	2	0	0
3	2	0.25	1	3	4	0
4	3	0.125	2	4	5	6
5	4	0.0625	3	5	7	8
6	5	0.03125	3	7	9	10

Error Montecarlo

```
T = zeros(5,1);
for k = 3:7
    M = 10^k;
    [resultat, points] = metode_montecarlo(f,a,b, M);
    T(k-2) = resultat;
end
```

```
k = (3:7)';
M = (ones(5,1)*10).^k;
T = [k M T error_absoluto(log(2),T) error_relativo(log(2),T) xifres_significatives_correctes(lo
t = array2table(T, 'VariableNames', {'k', 'M', 'va_ln2', 'errAbs', 'errRel', 'xifres'})
```

t = 5×6 table

	k	M	va_ln2	errAbs	errRel	xifres
1	3	1000	0.697	0.003852...	0.005558...	1
2	4	10000	0.6854	0.007747...	0.011176...	1
3	5	100000	0.69411	0.000962...	0.001389...	2
4	6	1000000	0.692722	0.000425...	0.000613...	2
5	7	10000000	0.6931696	2.241944005...	3.234441498...	4

4.e

Como podemos ver para que el metodo de Montecarlo tenga el mismo numero de cifras significativas correctas que el metodo del Trapezio tenemos que tener una muestra de 10^5. En contra posición para obtener mismo numero de cifras significativas que Simspon, la muestra de Montecarlo tiene que ser muy grande.

```
function [T, trapezis] = regla_composta_trapezis_iterations(f, a, b, base, iterations)
    k = 0:(iterations-1);
    h = base.^k;
    trapezis = zeros(iterations,1);
    for i = 1:iterations
        sumatorio = f(a:h(i):b);
        sumatorio = sum(sumatorio(2:end-1));
        trapezis(i,1) = h(i)*(((f(a)+f(b))/2) + sumatorio);
    end
    integrate = integral(f,a,b);
    errAbs = abs(trapezis-integrate);
    T = [k' h' trapezis errAbs];
end
function [T, taula_romberg] = metode_romberg(f,a,b,iterations, N)
    base = 1/2;
    [~, trapezis] = regla_composta_trapezis_iterations(f,a,b,base, iterations);
    taula_romberg = zeros(iterations,N);
    taula_romberg(:,1) = trapezis;
    for j = 2:N
        for i = j:iterations
            p = 4^(j-1);
            taula_romberg(i,j) = (p*taula_romberg(i,j-1)-taula_romberg(i-1,j-1))/(p-1);
        end
    end
    i = 1:iterations;
    h = base.^i;
    integrate = integral(f,a,b);
    error = abs(taula_romberg(:,N)-integrate);
    error(1:N-1,:) = 0;
    k = 0:(iterations-1);
    T = [k' h' taula_romberg error];
```

```
end
function [montecarlo, points] = metode_montecarlo(f,a,b, M)
        %Calcula el area en una region 1x1, cuya base se encuentra en el
        %eje de abscisas
        k=0;
        points = zeros(M,2);
        %Cambio de intervalo a un intervalo [0,1]
        f = @(t)f(a+t*(b-a));
        for n=1:M
            x = rand;
            y = rand;
            if y <= f(x)
                k = k+1;
                points(k,:) = [a+x*(b-a),y];
            end
        end
        points = points(1:k,:);
        %Como hemos hecho un cambio d intervalo hemos de multiplicar el
        %resultado de la integral por (b-a)
        montecarlo = (b-a)*(k/M);
end
function [T, taula_richardson] = extrapolation_richardson(f,point,iterations, N)
    base = 1/2;
    i = 1:iterations;
    h = base.^i;
    taula_richardson = zeros(iterations,N);
     for k = 1:iterations
        taula_richardson(k,1) = (f(point +h(k)) - f(point-h(k)))./(2*h(k));
     end
     for j = 2:N
         for i = j:iterations
             p = 4^(j-1);
             taula_richardson(i,j) = (p*taula_richardson(i,j-1)-taula_richardson(i-1,j-1))/(p-1
         end
     end
    syms x
    df = diff(f,x);
    dfx = double(df(point));
    error = abs(dfx-taula_richardson(:,N));
    error(1:N-1,:) = 0;
    T = [h' taula_richardson error];
end


function [coeficientes] = polinomio_interpolador_minimos_cuadrados(x, y, grado)
    A = ones(length(x),grado+1);
    i = grado;
    while(i  > 0)
        A(:,((grado+1)-i)) = (x').^i;
        i = i-1;
    end
    b = y';
    coeficientes  = minimos_cuadrados(A,b)';
end
```

```
function [x] = minimos_cuadrados(A,b)
    %A'*Ax=A'b
    C = A'*A;
    d = A'*b;
    x = C\d;
end
function [xifres_decimals_correctes] =  xifres_decimals_correctes(valor_exacto, valor_aproximad
    errorabsoluto = error_absoluto(valor_exacto, valor_aproximado);
    xifres_decimals_correctes = floor(abs(log10(errorabsoluto/0.5)));
end
function [xifres_significatives_correctes] = xifres_significatives_correctes(valor_exacto, valo
    errorrelativo = error_relativo(valor_exacto, valor_aproximado);
    xifres_significatives_correctes = floor(abs(log10(errorrelativo/0.5)));
end
function [error_absoluto] = error_absoluto(valor_exacto, valor_aproximado)
    error_absoluto = abs(valor_exacto-valor_aproximado);
end
function [error_relativo] = error_relativo(valor_exacto, valor_aproximado)
    error_relativo = abs(valor_exacto-valor_aproximado)./abs(valor_exacto);
end
```