

GUILLERMO HOYO BRAVO - ADRIÁN RUBIO PINTADO

In [23]:

```
import numpy as np
import matplotlib.pyplot as plt
import math
```

APARTADO 1

Cuestiones

1). Dado que el redondeo en base 10 de la suma o producto de dos números es $\pm 0,5 * \varepsilon$, donde ε es el valor la última cifra significativa y que el dígito eliminado es aleatorio ¿podemos suponer que el redondeo es una variable aleatoria uniforme?

Si, dado que el último dígito eliminado es una variable aleatoria y el redondeo depende de este podemos suponer que también es una variable uniforme aleatoria.

2) Si dibujo la gráfica de función de densidad del error por redondeo ¿Como debería ser dicha gráfica?

Debería ser un plano o una nube de puntos con los puntos medianamente alineados. Esto se debe a que la función de densidad de una variable aleatoria se comporta prediciendo comportamiento para el que una variable aleatoria tomará el valor X. El error como se ha comentado anteriormente es una variable aleatoria. Por lo que tomará valores cercanos a un punto a veces mayores y otras veces menores de manera uniforme.

3) Si asumimos que el error de redondeo es una variable aleatoria uniforme entre $-0,5 * \varepsilon$ y $\pm 0,5 * \varepsilon$ ¿Cuál debería ser error absoluto promedio de la suma (o el producto) de un número elevado de números en coma flotante?, da una respuesta razonada.

El error absoluto de la suma, es la suma de los diferentes errores. De tal manera que si se repite este proceso reiteradamente, como los errores varían entre $-0,5\varepsilon$ y $+0,5\varepsilon$, aparece una media de 0, ya que se compensan los errores.

Ejercicio 1

1a) Implementad en Python la siguiente función racional.

In [36]:

```
def polinomial1(x):

    numerator = 4*pow(x,4) - 59*pow(x,3) + 324*pow(x,2) - 751*x + 622
    denominator = pow(x,4) - 14*pow(x,3) + 72*pow(x,2) - 151*x + 112

    #Answer
    x = numerator/denominator

    return x
```

```
In [37]: i_array = np.arange(801) #i values

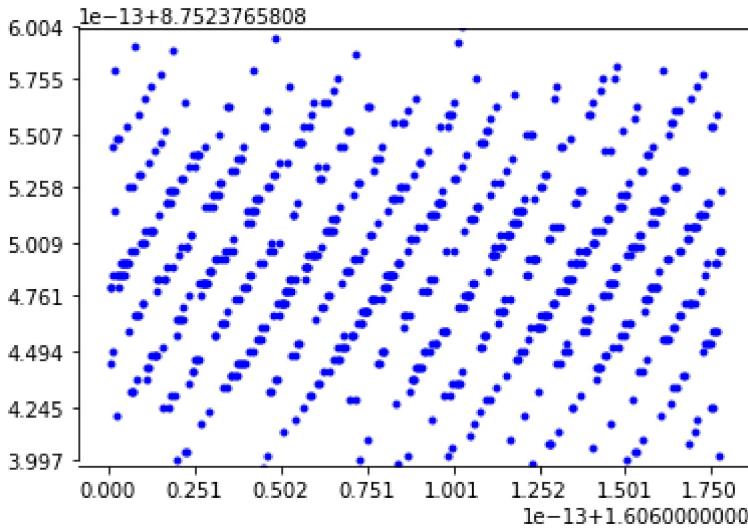
def x_value(i):
    return 1.606 + pow(2, -52)*i

x_array = [x_value(i) for i in i_array]
#print('x_array', x_array)

#apply
y_array = [polinomial1(x) for x in x_array]

#Plot
plt.plot(x_array,y_array, 'b.')
plt.ylim(8.7523765807784,8.7523765807786)
```

Out[37]: (8.7523765807784, 8.7523765807786)



¿Sale una figura continua?

No, se puede ver claramente que aparecen una serie de puntos con mismos valores para el eje vertical, incluso algunos superpuestos, lo que viola por completo la definición de continuidad.

¿Por qué?

Porque se ve que los puntos están dispersos con un patrón espaciado y repitiendo valores en vez de unidos de manera continua.

¿Puedes explicar el patrón que sale?

Sale un hiperplano. Un hiperplano divide el espacio muestra, por ejemplo, en un espacio de una dimensión (una recta), el hiperplano es de una dimensión, siendo un punto que divide la recta en dos. En un espacio de dos dimensiones, donde existe un plano, este puede ser dividido en dos por un hiperplano con forma de recta, y así sucesivamente. En un espacio de 4 se divide por un plano.

¿Qué consecuencias puedes sacar sobre el redondeo?

Que no es aleatorio debido a que aparece un patrón visible. Esperábamos una gráfica continua. Observamos como de manera empírica, a causa del redondeo, el error cambia de distribución,

por lo que no lo podemos considerar una variable aleatoria auténtica.

1b) Aplicación de la regla de Horner:

In [26]:

```
def horner(x):
    numerator = (622 + x * (-751 + x * (324 + x * (-59 + 4 * x))))
    denominator = (112 + x * (-151 + x * (72 + x * (-14 + x)))))

    #Answer
    x = numerator/denominator

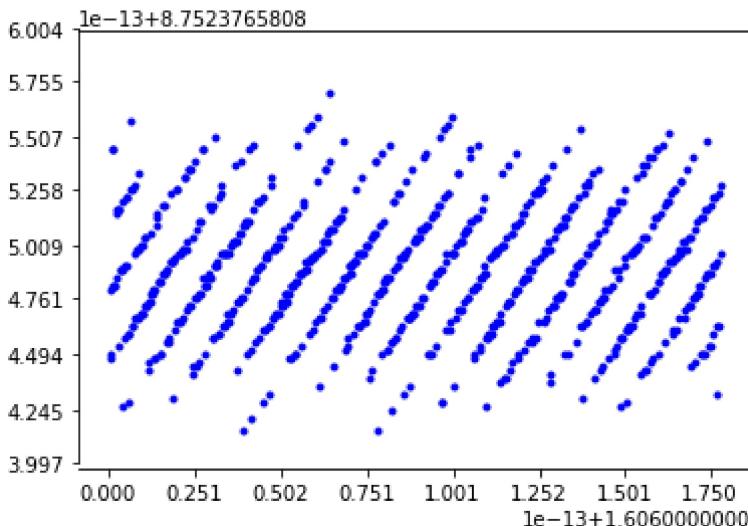
    return x
```

In [27]:

```
y_array_horner = [horner(x) for x in x_array]

#Plot
plt.plot(x_array,y_array_horner,'b.')
plt.ylim(8.7523765807784,8.7523765807786)
```

Out[27]: (8.7523765807784, 8.7523765807786)



¿Observas alguna diferencia con la gráfica anterior?

Sí, los puntos pertenecientes a cada línea horizontal ahora están menos dispersos entre sí, quedando mucho más juntos entre sí

1c) Un poquito de análisis previo

In [28]:

```
def analysis(x):
    numerator = 3 * (x - 2) * (pow(x - 5, 2) + 4)
    denominator = x + ( pow((x - 2), 2) * ((pow(x - 5, 2) + 3)) )

    #Answer
    x = 4 - (numerator/denominator)

    return x
```

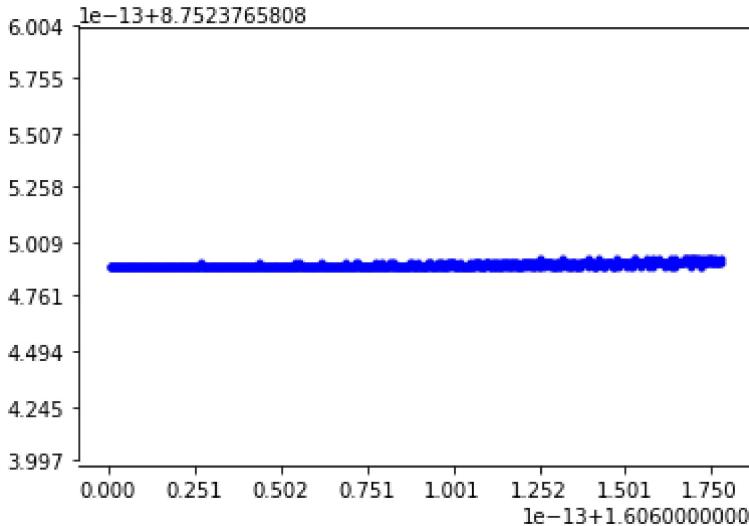
In [29]:

```
y_array_analysis = [analysis(x) for x in x_array]

#Plot
```

```
plt.plot(x_array,y_array_analysis,'b.')
plt.ylim(8.7523765807784,8.7523765807786)
```

Out[29]: (8.7523765807784, 8.7523765807786)



¿Observas alguna diferencia con las gráficas anteriores?

Si, los puntos están casi completamente alineados en una recta. El intervalo en el eje de la altura de la gráfica de esta recta está comprendido entre los valores: [4.761, 5.009].

Cuestiones

1) ¿Las tres funciones que hemos pintado son la misma función, solo que escrita de diferente manera? Da una respuesta razonada.

Las tres funciones son idénticas pero generan los números de manera diferente. Esto crea que el error generado en cada una de ellas sea diferente, y por lo tanto en la última realizada muchísimo menor que en la primera, incluso que en la segunda. Según como se realicen los cálculos los errores se van añadiendo e incrementando.

2) ¿Podemos afirmar ahora que la distribución del error por redondeo es una variable aleatoria uniforme?

No, no podemos afirmarlo. Como hemos observado se generan patrones en vez de ser valores aleatorios.

3) Comenta los resultados obtenidos.

Se obtienen resultados de peor a mejor formando hiperplanos los dos primeros ejercicios, y una recta el tercero. Esto es por lo que se acaba de mencionar de que el error en verdad no es una variable aleatoria.

Ejercicio Opcional

(solo para los que les molestan los auténticos fenómenos numéricos paranormales). Dibujad ahora las mismas gráficas para valores de $x = 2.4 + 2 - 52 i$ con $i=0,1,\dots,800$, para los límites de la gráfica usad `plt.ylim(.7407108239094,.74071082390965)` ¿Qué cambio has observado? ¿Podrías explicar por qué ha cambiado la dirección del patrón?

```
In [30]: def polinomial1(x):
    numerator = 4*pow(x,4) - 59*pow(x,3) + 324*pow(x,2) - 751*x + 622
    denominator = pow(x,4) - 14*pow(x,3) + 72*pow(x,2) - 151*x + 112

    #Answer
    x = numerator/denominator

    return x
```

```
In [31]: i_array = np.arange(801) #i values
#print('i_array', i_array)

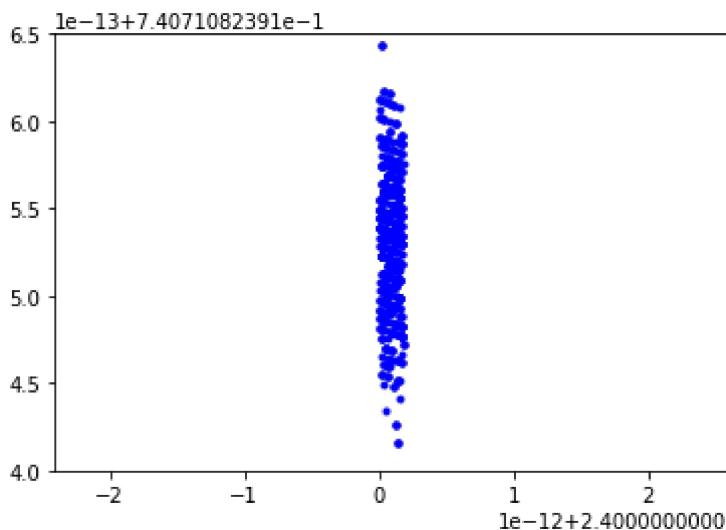
def x_value(i):
    return 2.4 + pow(2, -52)*i

x_array = [x_value(i) for i in i_array]
#print('x_array', x_array)

#apply
y_array = [polinomial1(x) for x in x_array]

#Plot
plt.plot(x_array,y_array, 'b.')
plt.ylim(.7407108239094,.74071082390965)
```

Out[31]: (0.7407108239094, 0.74071082390965)



```
In [32]: def horner(x):
    numerator = (622 + x * (-751 + x * (324 + x * (-59 + 4 * x))))
    denominator = (112 + x * (-151 + x * (72 + x * (-14 + x)))) 

    #Answer
    x = numerator/denominator

    return x
```

```
In [33]: i_array = np.arange(801) #i values
#print('i_array', i_array)

def x_value(i):
    return 2.4 + pow(2, -52)*i

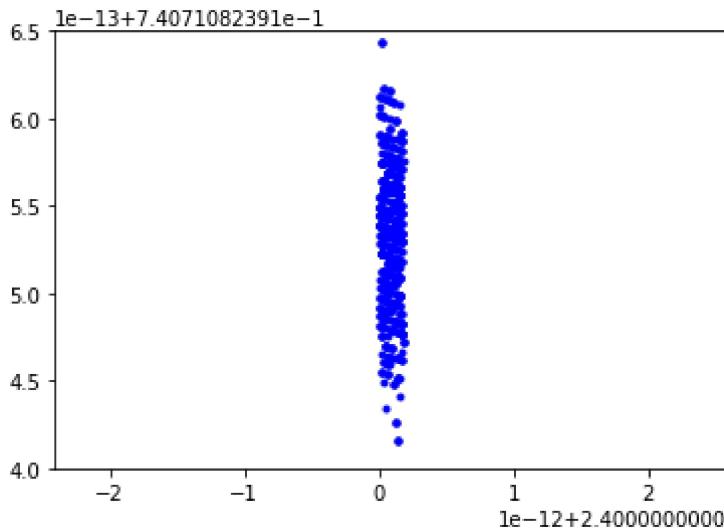
x_array = [x_value(i) for i in i_array]
```

```
#print('x_array', x_array)

#apply
y_array = [polinomial1(x) for x in x_array]

#Plot
plt.plot(x_array,y_array,'b.')
plt.ylim(.7407108239094,.74071082390965)
```

Out[33]: (0.7407108239094, 0.74071082390965)



In [34]:

```
def analysis(x):
    numerator = 3 * (x - 2) * (pow(x - 5, 2) + 4)
    denominator = x + (pow((x - 2), 2) * ((pow(x - 5, 2) + 3)) )

    #Answer
    x = 4 - (numerator/denominator)

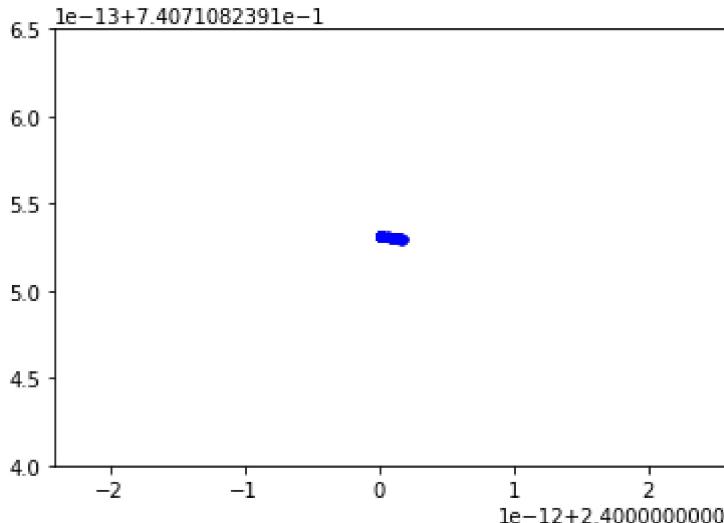
    return x
```

In [35]:

```
y_array_analysis = [analysis(x) for x in x_array]

#Plot
plt.plot(x_array,y_array_analysis,'b.')
plt.ylim(.7407108239094,.74071082390965)
```

Out[35]: (0.7407108239094, 0.74071082390965)



¿Qué cambio has observado? ¿Podrías explicar por qué ha cambiado la dirección del patrón?

Se observa que el patron ahora se condensa en una recta vertical, así como que el error es menor. Suponemos que este cambio en la dirección del patron se debe a que el numero 2,4 es mucho mas facil de computar y de representar, o arrastra menor error su representación que el anterior 1,606.

APARTADO 2: APROXIMACIÓN DE FUNCIONES:

Preguntas

1. ¿Dado un conjunto de n puntos, existe siempre un polinomio de grado m < n-1 que pase por dichos puntos?

No, no siempre. Sería cierto si la condición fuera que el grado del polinomio fuera de grado $m \leq n-1$. Sin embargo, con dicha condición (estrictamente menor) requiere que el polinomio sea como mucho de grado $n-2$. Podemos verlo con un contraejemplo sencillo:

Dados 3 puntos, necesitariamos un polinomio de grado $m < 3 - 1 = 2$, es decir, de grado como mucho 1, o lo que es lo mismo, una recta, que pasara por los 3 puntos . Si los puntos no están alineados, es imposible encontrar una recta que pase por los 3 puntos simultaneamente.

Vemos que la afirmación del enunciado no es verdad para casos generales, ya que hemos encontrado un contraejemplo que la desmiente. Por lo que no, no existe siempre un polinomio de grado $m < n-1$ que pase por dichos puntos (n).

2. ¿Se te ocurre una manera en la cual el cálculo de los valores singulares de una matriz permita calcular un polinomio de regresión adecuado?

Sí. Tal y como comentamos en el ejercicio 3, podemos transformar el cálculo del polinomio de regresión en un problema de producto de matrices utilizando la ecuación normal.

Una matriz la podemos descomponer de la siguiente forma:

$$A = U * S * V^t$$

donde

A: es LA matriz mxn

U: es una matriz unitaria mxm

V: es una matriz unitaria nxn

valores singulares en la diagonal

S: es una matriz diagonal con los

Es decir, una vez calculados los valores singulares, los utilizamos para construir la matriz diagonal S. Una vez calculadas U y V_trapuesta, podemos sustituir por la matriz A. Si además

multiplicamos por la izquierda cada término por la pseudoinversa de A: tenemos:

$$A_{pseudo} * A * c = A_{pseudo} * y$$

Aplicando la definición de pseudoinversa:

$$A_{pseudo} = V * S^t * U^t$$

Obtenemos la ecuación que nos interesa

$$c = V * z$$

Siendo el vector

$$z = \begin{pmatrix} \frac{c_1}{s_1} \\ \frac{c_2}{s_2} \\ \frac{c_3}{s_3} \end{pmatrix}$$

, donde los valores si corresponden a los valores singulares de la matriz A y los valores c_i corresponden a los tres primeros valores del vector obtenido al multiplicar la matriz U_traspuesta por y = (y1, y2, y3, y4, y5).

Es decir, el cálculo de los valores singulares nos vale para el cálculo del polinomio de regresión que queremos.

3. ¿Como podrías transformar el problema de encontrar un polinomio de regresión en un problema de producto de matrices?

Dado que en regresión lineal queremos encontrar un recta que se ajuste lo mejor posible a los puntos dados, dicha recta tiene una ecuación tal que así:

$$y = c_1 * x + c_2$$

Para que la recta pase por todos ellos, establecemos un conjunto de ecuaciones para cada punto de la forma:

$$y_i = c_1 * x_i + c_2$$

Como tenemos un sistema de ecuaciones podemos expresarlo en forma matriz de la siguiente manera:

$$Ac = y$$

donde

$$A = \begin{pmatrix} X_1 & 1 \\ \dots & \dots \\ X_N & 1 \end{pmatrix}$$

$$c = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$$

$$y = \begin{pmatrix} y_1 \\ \dots \\ y_N \end{pmatrix}$$

Obteniendo un sistema sobredeterminado. Por eso deseamos encontrar una recta que minimice las desviaciones a cada punto (error e), tal que:

$$r = y - A * c$$

Minimizando el cuadrado de la norma 2 de r, obtenemos la ECUACIÓN NORMAL:

$$(A^t * A) * c = A^t * y$$

Es decir, hemos conseguido **transformar el problema de encontrar un polinomio de regresión en un problema de producto de matrices**

4. ¿Crees que, además, es posible calcular el error de regresión a partir de los valores singulares?

Sí, se puede demostrar que el error cometido con la regresión obtenida es

$$E = \sqrt{(c4)^2 + (c5)^2}$$

(Tras consultar varias fuentes, dada a la extensión de la demostración matemática no la hemos incluido finalmente en el notebook.)

APARTADO 1

1 a) Vamos a escribir el problema de regresión como un problema de interpolación sobredimensionado asumiendo que el polinomio P2(x) pasa por todos los puntos x_i, y_i . Para ello escribiremos el problema en la forma $Xa = y$ donde X es una matriz 5×3 , a es un vector 3×1 e y es un vector 5×1 .

Planteamos el sistema sobredimensionado de 5 ecuaciones con 3 incógnitas

In [13]:

```
#Xa = y
x_1 = 0
x_2 = 0.25
x_3 = 0.5
x_4 = 0.75
x_5 = 1

X = np.array([
    [x_1**2, x_1, 1],
    [x_2**2, x_2, 1],
    [x_3**2, x_3, 1],
    [x_4**2, x_4, 1],
    [x_5, x_5, 1]
])

y = np.array([
    [1],
    [1.2840],
    [1.6487],
    [2.1170],
    [2.7183]
])
```

```
# ECUACIÓN NORMAL
# Xt *X * a = Xt*y
```

1b) Calculad la descomposición en valores singulares $X = U S Vt$ de la matriz X que habéis encontrado en el apartado anterior. Utilizad los apuntes de clase para obtener los valores singulares de X y para construir las matrices U de 5 x5, S de 5 x 3 y V 3x3.

Utilizamos las numpy para el cálculo de la descomposición de la matriz.

In [3]:

```
u,s,vh=np.linalg.svd(X)
smat = np.zeros((5, 3))
smat[:3, :3] = np.diag(s)
#print(s)
#print(smat)
print('¿All close?:',np.allclose(X, np.dot(u, np.dot(smat, vh))))
```

¿All close?: True

Imprimimos las matrices obtenidas:

In [4]:

```
print('Matriz U:\n', u)
print('\n')
print('Matriz S:\n', s)
print('\n')
print('Matriz V_traspuesta:\n', vh)
print('\n')
```

Matriz U:

```
[[ -0.29454913  0.63267484  0.63140925  0.06554652 -0.33164645]
 [-0.346616   0.45500901 -0.21036331  0.07188383  0.78955946]
 [-0.41593037  0.19422733 -0.52439676 -0.60893063 -0.37879968]
 [-0.50249224 -0.14967019 -0.3106911   0.74002367 -0.28449322]
 [-0.60630161 -0.57668356  0.43075367 -0.2685234   0.20537989]]
```

Matriz S:

```
[ 2.71168512  0.93707467  0.16268803]
```

Matriz V_traspuesta:

```
[[ -0.37415833 -0.4712162  -0.7987245 ]
 [-0.62308604 -0.51017309  0.59286357]
 [ 0.68685467 -0.71949893  0.10272272]]
```

APARTADO 2: $a = Vz$

Se puede demostrar (de hecho, no es muy difícil), que los valores $a = (a_0, a_1 y a_2)$ del polinomio que buscamos se pueden obtener mediante el siguiente producto de matrices

$$a = Vz$$

El vector $z = (c_1/s_1, c_2/s_2, c_3/s_3)$, donde los valores si corresponden a los valores singulares de la matriz A y los valores ci corresponden a los tres primeros valores del vector obtenido al multiplicar la matriz Ut por $y = (y_1, y_2, y_3, y_4, y_5)$.

Calculamos a por medio de $a = Vz$

In [5]:

```
# a = Vz
```

```
#CALCULAMOS EL VECTOR Z = (c1/s1, c2/s2, c3/s3)

# Ya tenemos los valores singulares en s
# Obtenemos ahora los valores de c1,c2 y c3
c_values = np.dot(u.T,y)
#print(c_values)

z = np.array([ c_values[0]/s[0] ,
               c_values[1]/s[1] ,
               c_values[2]/s[2] ])

# Calculamos las constantes a del polinomio de regresión que queremos calcular media
a = np.dot(vh.T,z)
print('Constantes a:\n', a)
```

Constantes a:

```
[[0.84365714]
 [0.86418286]
 [1.00513714]]
```

Pintamos los puntos dados junto con el polinomio de regresión

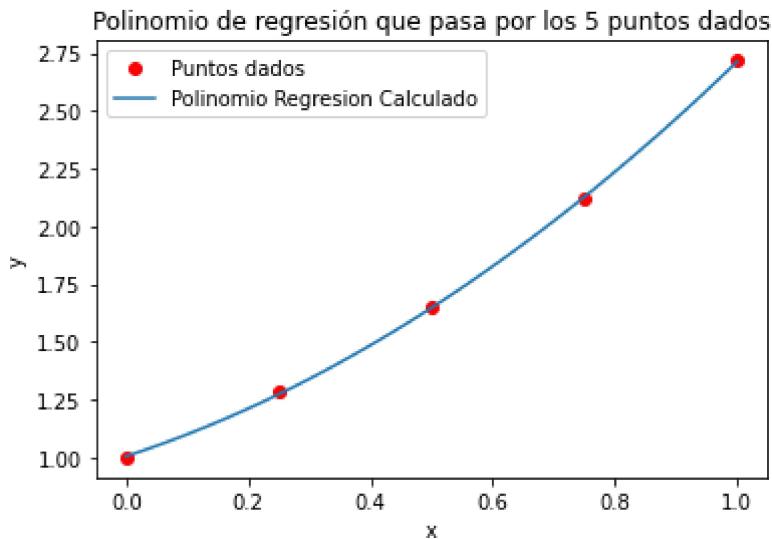
In [6]:

```
#Puntos por los que tiene que pasar el polinomio
plt.plot([x_1,x_2,x_3,x_4,x_5], y.flatten().tolist(), 'ro', label = 'Puntos dados')

#Evaluamos el polinomio de regresión calculado evaluandolo en x = [0,1]
pregresion = lambda x: a[0] * x**2 + a[1] * x + a[2]
x_pregresion = np.linspace(0,1,100)
y_pregresion = [pregresion(i) for i in x_pregresion]
plt.plot(x_pregresion, y_pregresion )

plt.legend(['Puntos dados', 'Polinomio Regresion Calculado'])
plt.xlabel('x')
plt.ylabel('y')
plt.title('Polinomio de regresión que pasa por los 5 puntos dados')

plt.show()
```



Observamos como el polinomio de regresión pasa por los puntos dados tal y como esperábamos. Concluimos que es más efectivo computacionalmente calcular el polinomio de regresión(calculando los valores singulares de a) mediante el producto $a = Vz$, ya que nos ahorra tiempo de ejecución y tal y como hemos comprobado funciona.

APARTADO 3: Error de Regresión

Apartado 3: Se puede demostrar también (es más fácil aun) que el error cometido con la regresión obtenida para los valores es $E = \sqrt{c_4^2 + c_5^2}$, calculad el error y comparadlo con el cálculo manual del error (es decir, comparad el valor E con el valor obtenido al calcular $\sqrt{\sum (P2(xi) - yi)^2}$)

Calculamos el error cometido con los valores $E = \text{raiz_cuadrada}(c_4^2 + c_5^2)$

In [7]:

```
# Error cometido con Los valores E = raiz_cuadrada(c_4**2 + c_5**2)
error_regr = np.sqrt(c_values[3]**2 + c_values[4]**2)
print('Error cometido con la regresión', error_regr[0])
```

Error cometido con la regresión 0.016556949339433424

Y evaluamos el polinomio para los puntos dados en el eje x

In [8]:

```
#Evaluamos el polinomio para Los puntos dados en el eje x
x_values = [x_1,x_2,x_3,x_4, x_5]
y_values = [pregresion(i)[0] for i in x_values]
print('Valores del Polinomio de Regresión evaluado en los puntos dados:\n', y_values)
```

Valores del Polinomio de Regresión evaluado en los puntos dados:

[1.005137142857143, 1.2739114285714286, 1.6481428571428574, 2.127831428571429, 2.7129771428571434]

Y ahora calculamos el error del polinomio de regresion

In [9]:

```
#Error del polinomio de regresion
y_reales = y.flatten().tolist()
diferencias = np.subtract(y_values, y_reales)
#print(diferencias)
diferencias= [i**2 for i in diferencias]
error = np.sqrt(np.sum(diferencias ))
print('Error Manual del Polinomio de Regresión', error)
```

Error Manual del Polinomio de Regresión 0.01655694933943384

Comparativa de los errores

Hacemos ahora una comparativa de los errores para ver como de distantes son

In [10]:

```
print('Error cometido con la regresión', error_regr[0])
print('Error Manual del Polinomio de Regresión', error)
print('\n')
print('Diferencia de los errores calculados', error - error_regr[0])
```

Error cometido con la regresión 0.016556949339433424

Error Manual del Polinomio de Regresión 0.01655694933943384

Diferencia de los errores calculados 4.163336342344337e-16

Observamos como la diferencia del calculo del error es despreciable $4e-16$, que está cerca del epsilon de la máquina($2e-16$), por lo que podemos utilizar perfectamente la formula

$$E = \sqrt{(c_4)^2 + (c_5)^2}$$

para el cálculo del error en el polinomio de regresión por mínimos cuadrados, evitándonos tener que evaluar al polinomio para el cálculo del error.

EJERCICIO OPCIONAL

A) ¿Qué obtenemos cuando aplicamos el método de los apartados 1 y 2 a la construcción de un polinomio de grado 4?

Obtenemos un polinomio interpolador en vez de uno de regresión

B) Comprueba que, efectivamente el polinomio obtenido se corresponde en este caso a un polinomio de interpolación en lugar de a un polinomio de regresión.

Tenemos 5 puntos y queremos construir un polinomio de grado 4. Siempre que tenemos un número N de puntos y queremos construir un polinomio de grado N-1, sabemos por el teorema del polinomio interpolador, que ese polinomio existe y es único.

Es decir, planteamos un sistema con solución(y además única), por lo que deja de ser un problema de regresión y pasa a ser uno de interpolación.