ILI 285
Laboratorio #4

Alonso Sandoval Acevedo
asandova@alumnos.inf.utfsm.cl
201073011-5

Hernán Vargas Leighton
hvargas@alumnos.inf.utfsm.cl
201073009-3

2 de julio de 2014

1. Descripción del experimento

En primer lugar se ampliará una imagen utilizando las funciones interpoladoras, esperamos que el resultado sea optimo para los splines, pero creemos que con Newton obtendremos ruido debido al fenómeno de Runge.
En la segunda parte analizaremos los beneficios y contras al utilizar puntos de Chebyshev para aproximar una función. Observaremos que es lo que ocurre al aumentar los puntos óptimos utilizados, como también veremos como afecta la variación del parámetro α de la función al error de la interpolación.

2. Desarrollo

2.1. Image Resizing

a) La estrategia utilizada para agrandar las imágenes consiste en aplicar una función de interpolación para filas y luego columnas, en este caso se utilizó la función `interpol1d` de `scipy.interpolate` para splines cúbicos y, para las diferencias divididas de Newton, se modifico en menor medida el código de Ernesto P. Adorio, obtenido en la página web `http://adorio-research.org`.
El algoritmo toma los pixeles de la imagen original y los distribuye equis-espaciados en la nueva imagen. En los puntos en los cuales no se tiene información, se utiliza la función elegida para interpolar su valor. Este proceso se hace primero para todas las lineas y, una vez terminado, tendremos una imagen expandida solo en una de sus dimensiones, es decir $A^{m\times n} \to A^{m\times n\cdot L}$ siendo L la razón de expansión. En este caso, con splines cúbicos y expandiendo a 5X tenemos:

Figura 1: Expansión de una dimensión

Para una imagen $A^{m\times n}$ este proceso conlleva el calculo de m funciones interpoladoras, además debemos evaluar dichas funciones por cada uno de los pixeles de la nueva imagen, es decir, $n \cdot L$ veces cada función, $n \cdot L \cdot m$ evaluaciones en total.
Tomando ésta como la nueva imagen a expandir, nos basta repetir el proceso pero ahora para las columnas y con ello conseguiremos la imagen expandida final.
En el proceso de expansión de de $A^{m\times n\cdot L} \to A^{L(m\times n)}$ tenemos que calcular $n \cdot L$ funciones interpoladoras y evaluar cada una de ella en $m \cdot L$ puntos, en total $m \cdot n \cdot L^2$ evaluaciones.

Las imágenes resultantes serán:

Figura 2: Splines cúbicos

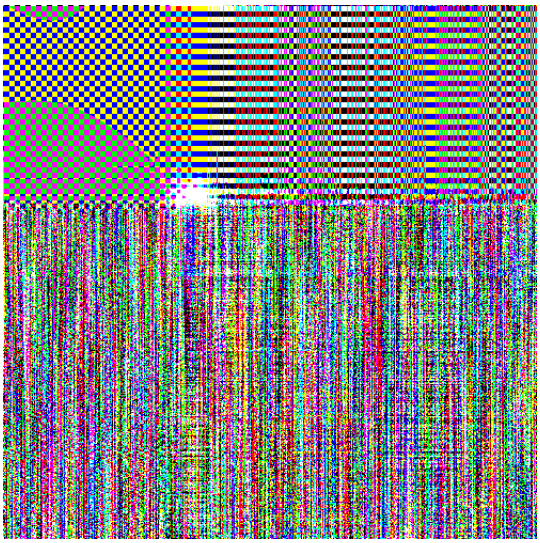


Figura 3: Diferencias divididas de Newton

Como podemos ver que el proceso con splines cúbicos da los resultados esperados, pero para Newton solo obtenemos ruido. Creemos que este problema sucede debido al fenómeno de Runge, mientras más puntos tenemos, el error de la interpolación entre ellos aumentará considerablemente.
Nuestro algoritmo esta escrito de manera que si encuentra un valor mayor a 255 o menor a 0 simplemente escriba en la matriz un 255 o 0 respectivamente, así se podrá siempre generar una imagen, es por ello que, a pesar de que los valores de interpolación de Newton deberían salir del rango de representación `RGB`, podemos ver este ruido.
El gran problema que encontramos a la hora de agrandar las imágenes fue el tiempo que demora el algoritmo en hacer la ampliación. Para analizar este factor primero debemos recordar que el proceso completo necesitará:

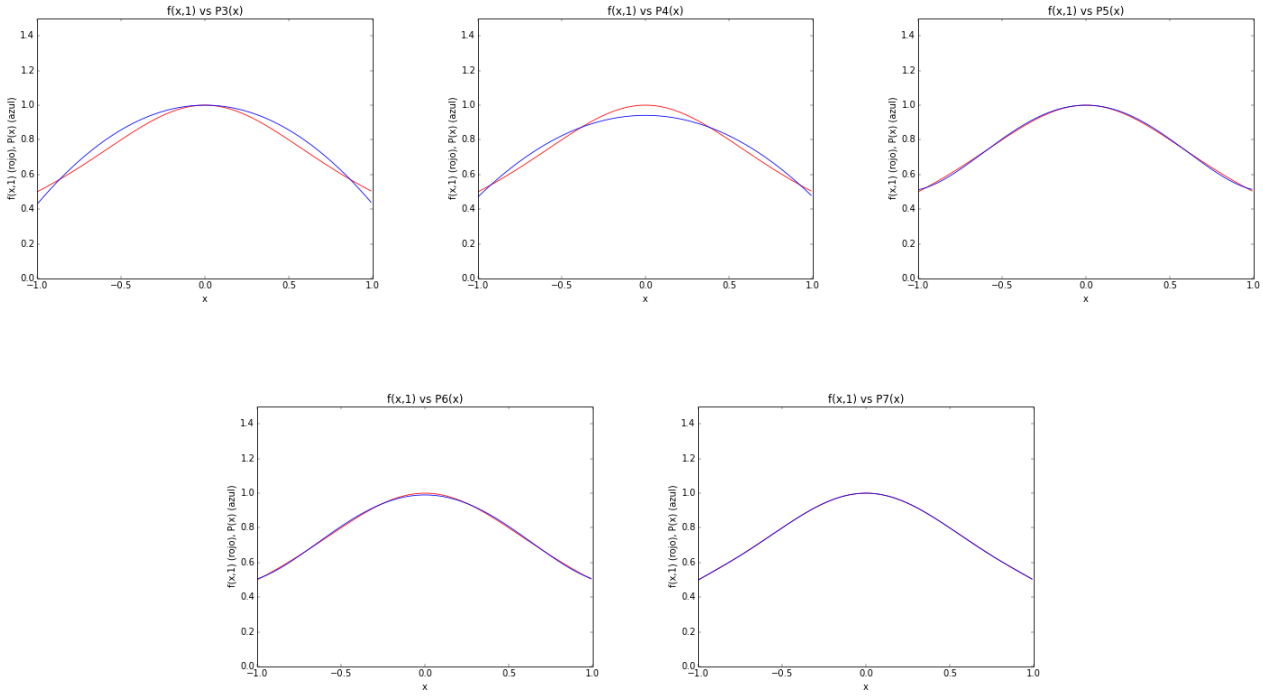
- $m + n \cdot L$ funciones interpoladoras.
- $n \cdot m \cdot (L + L^2)$ evaluaciones de dichas funciones.

Sin considerar el tiempo que se gasta en las asignaciones de memoria y cálculos menores.
Para la imagen suministrada tenemos que la resolución es de 100×100 pixeles y queremos una ampliación `5X`, es decir $m = 100$, $n = 100$ y $L = 5$, reemplazando en las formulas tenemos que necesitaremos calcular 600 funciones interpoladoras y evaluar 350000 veces en total. Para la interpolación con splines cúbicos tenemos que, en el computador que trabajamos*, para obtener una función de interpolación se demora aproximadamente 0,013 segundos, mientras que para evaluarla demoró cerca de 0,00015 segundos. Con estos tiempos tenemos que las obtención de las funciones interpoladoras demorará cerca de 8 segundos y la evaluación de estas tomará 53 segundos aproximadamente, con ello concluimos que el algoritmo demorará al menos 61 segundos, y ello sin tener en cuenta el resto de los cálculos y el trabajo de memoria.
Creemos que esta es una de las principales razones por las cuales el algoritmo es lento. Para solucionar este problema podríamos pensar en algún algoritmo que interpole ambas dimensiones al mismo tiempo, o mejor aún, aprovechando que el calculo de las funciones interpoladoras y su aplicación en la respectiva fila/columna es independiente de las demás, podríamos hacer el proceso utilizando multi-procesamiento, de esta manera obtener y evaluar paralelamente todas las funciones interpoladoras de las filas y luego hacer lo mismo para las columnas.

*Dell-xps-15: Intel Core i7 CPU Q 740 @ 1.734GHz, 6GB 1333Mhz DDR3

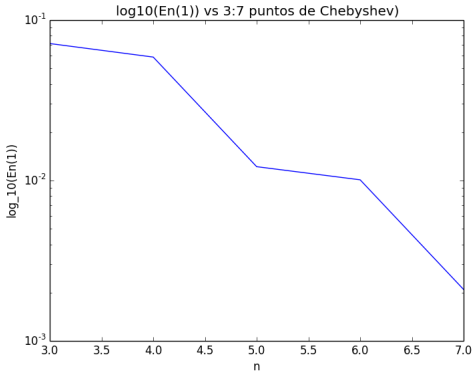
2.2. Puntos de Chebyshev

a) Gráficos de $f(x,1)$ y $P_n(x)$



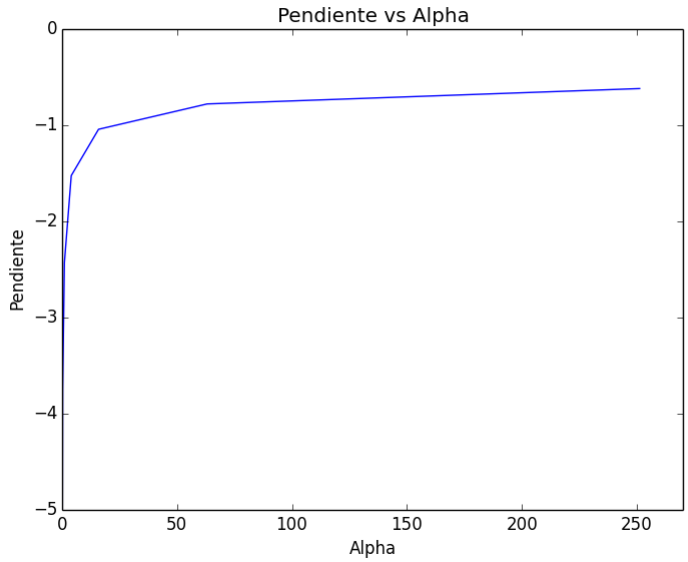
Para la interpolación utilizamos el método diferencias divididas de Newton, programado por nosotros, en conjunto a los n puntos de Chebyshev con los cuales evaluamos en $f(x,1)$ y generamos los datos que nos permiten interpolar la función.
Es evidente en los gráficos que al aumentar el número de puntos de Chebyshev mejora la aproximación de la función interpoladora(Azul).

b) Gráfico del error en escala logarítmica versus el número de puntos de Chebyshev, $n = 3:7$.



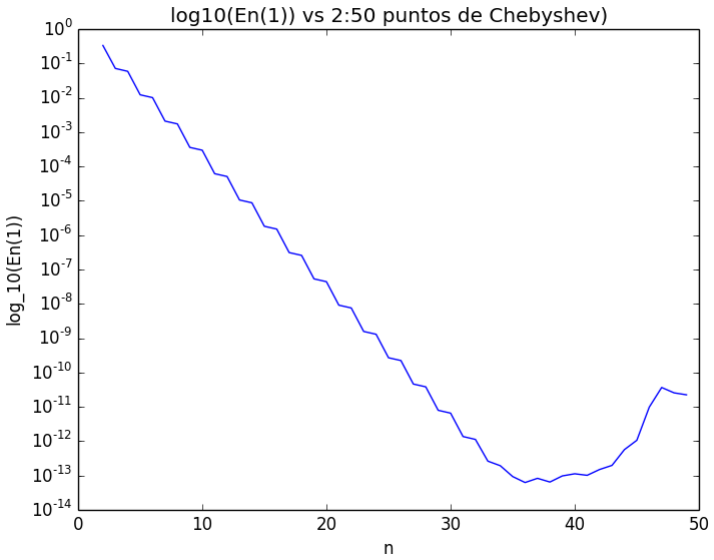
El gráfico es condescendiente con los resultados del punto anterior, a medida que se aumentan los puntos de Chebyshev, el error entre la función original y su interpolación disminuye, es decir mejora la aproximación de $P(x)$ a $f(x,1)$. Notar que la pendiente es negativa a simple vista, un cálculo aproximado de la pendiente (mediante regresion lineal en python) nos da un valor de: $s : -2,43$

c) Gráfico de la pendiente del error versus el α de la función $f(x, \alpha)$



Según nuestro gráfico es mejor un α pequeño, dado que la pendiente del gráfico es más negativa y por lo tanto, el error disminuye más rapido a medida que aumentamos los puntos de Chebyshev (hasta una cierta cantidad). A mayor valor de α, menor es la pendiente del error, y por lo tanto, los puntos de Chebyshev se hacen menos significativos (La pendiente del error tiende a 0, por tanto el error practicamente no varia para α grande).

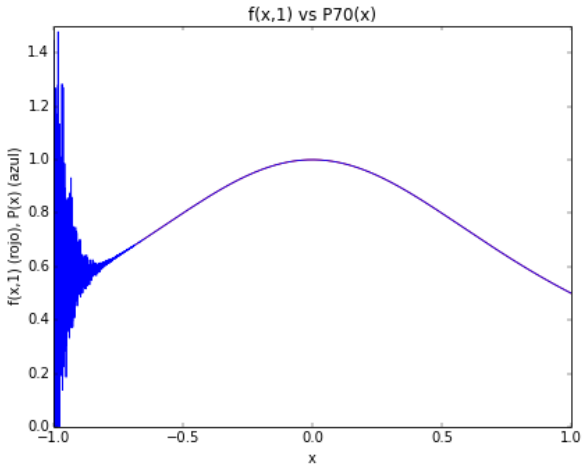
d) Gráfico del error en escala logarítmica versus el número de puntos de Chebyshev, $n = 2 : 50$.



El gráfico del error se comporta normalmente hasta valores cercanos a 10^{-14}, aproximadamente 40 puntos de Chebyshev, posteriormente los valores del error comienzan a aumentar nuevamente, ¿por qué?, pensamos que se debe a la representación computacional, por ejemplo, para valores muy pequeños de Chebyshev, pueden ocurrir errores de cancelación o cifras significativas en el algoritmo diferencias divididas de Newton, lo cual arruina la interpolación.

3. Conclusiones

Con respecto al proceso de agrandar imágenes podemos concluir que si bien nuestro algoritmo logra el resultado esperado en el caso de splines cúbicos se demora demasiado, por lo cual, no debe ser la manera en la cual se hace este proceso actualmente. Por otro lado, vemos que Newton no es un buen método de interpolación para las imágenes debido al fenómeno de Runge.
Para la sección dos concluimos que a mayor cantidad de puntos de Chebyshev mejor es la interpolación, sin embargo, sobre cierta cantidad, 40 en nuestro caso, el error deja de disminuir y comenzará a crecer, creemos que uno de los causantes de este fenómeno son valores computacionales muy pequeños en los puntos de Chebyshev los que hacen divergir a la función, afectando al algoritmo de D.D de Newton.
Otro enfoque es el fenómeno de Runge, el cual podemos ver en el siguiente gráfico:



Por ultimo, la interpolación también dependerá de la función, en nuestro caso, a mayor α para $f(x, \alpha)$ menor es el impacto de los puntos de Chebyshev, por lo que la pendiente del error decrece.

4. Fuentes

- **scipy - interp1d:** Sobre la función utilizada para interpolar con splines cúbicos, enlace: http://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.interp1d.html
- **Adorio-research.org - Newton interpolating polynomial for approximations:** Fuente de la función utilizada para interpolar con diferencias divididas de Newton en la primera parte, enlace: http://adorio-research.org/wordpress/?p=11165

5. Anexo

../Codigos/Laboratorio4.py

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
from PIL import Image
import math
from numpy import *
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d as spline

#### Parte 1: Agrandando imágenes. ####
IMAGE_PATH = './homero.png'

#Diferencias divididas de newton
def ddNewton(x, y):
    #Código modificado de: http://adorio-research.org/wordpress/?p=11165
    #Obteniendo coeficientes:
    c = y[:]
    n = len(x)
    for i in range(1, n):
        for j in range(n-1, i-1, -1):
            c[j] = (c[j] - c[j-1])/(x[j]-x[j-i])
    #Creando la función:
    def _lambda(xpoint):
        val = c[0]
        factor = 1.0
        for i in range(1, n):
            factor *= (xpoint - x[i-1])
            val += (c[i] * factor)
        return val
    return _lambda

#Clase encargada de tratar con las imágenes.
class MyImage():
    # Abre la imagen y la separa en colores
    def __init__(self, path, size = None):
        img = Image.open(path)
        data = array(img)
        if size == None: self.X, self.Y = img.size
        else: self.X, self.Y = size
        self.data = {'R': [], 'G': [], 'B': []}
        for x in range(self.X):
            for y in range(self.Y):
                for key, num in [('R',0), ('G',1), ('B', 2)]:
                    self.data[key].append( data[x,y][num] )

    # Agranda la imagen, recibe un modo (spline o newton) y aplica dicho
    # algoritmo para interpolar los nuevos valores. Luego guarda la imagen.
    def resize(self, mode, ratio, name):
        if mode == 'spline': f_mode = lambda x,y: spline(x,y, kind = 'cubic')
        elif mode == 'newton': f_mode = ddNewton
        else: raise ValueError('modos permitidos: spline, newton')
        new_x, new_y = self.X * ratio, self.Y * ratio
        ## Para filas:
```

```
        dic_f = {'R':{}, 'G':{}, 'B':{}}
        p_x = [k*(new_x/(self.X -1.0)) for k in range(self.X)]
        #Buscamos funciones:
        for y in range(self.Y):
            for c in ['R','G','B']:
                dic_f[c][y] = f_mode(p_x, self.data[c][y*self.X:(y+1)*self.X])
        #Creamos nueva matriz:
        new_data = {'R':[], 'G':[], 'B':[]}
        for y in range(self.Y):
            for x in range(new_x):
                for c in ['R','G','B']:
                    new_data[c].append(dic_f[c][y](x))

        #Ahora para las columnas con la imagen extendida.
        dic_f = {'R':{}, 'G':{}, 'B':{}}
        p_y = [k*(new_y/(self.Y -1.0)) for k in range(self.Y)]
        #Buscamos funciones:
        for i in range(new_x):
            for c in ['R','G','B']:
                dic_f[c][i] = f_mode(p_y, [new_data[c][new_x*j + i]
                                           for j in range(self.Y)])
        #Evaluamos:
        new_data2 = {'R':[], 'G':[], 'B':[]}
        for x in range(new_x):
            for y in range(new_y):
                for c in ['R','G','B']:
                    try: value = dic_f[c][y](x)
                    except: value = 0
                    if value < 0: value = 0
                    elif value>255: value = 255
                    new_data2[c].append(int(value))
        # Creamos la imagen y guardamos:
        img = Image.new('RGB', (new_x,new_y))
        img.putdata(zip(new_data2['R'], new_data2['G'], new_data2['B']))
        img.show()
        img.save(name)

img = MyImage(IMAGE_PATH, (100,100))
# Des-comentar las siguientes lineas para crear las imágenes, Nota: puede tomar
# mucho tiempo, en especial Newton.
#img.resize('spline',5,'../Informe/Graficos/5xhomero_spline.png')
#img.resize('newton',5,'../Informe/Graficos/5xhomero_newton.png')

################################################################################

#### Parte 2: Puntos de Chebysev ####

#Función para la tarea
f1 = lambda x,y: 1./(y + x**2)

# Retorna los puntos óptimos de Chebyshev
def chebPoints(n):
    xi = []
    for i in range(1,n+1):
        a = (2*i-1)*math.pi
```

```
        b = 2*n
        xi.append(math.cos(a/b))
    return asarray(xi)

#coNewton: Coeficientes de Newton para D.D.Newton
#Recibe: los puntos de Chebyshev
#Retorna: Coeficientes de D.D. de Newton (f[x1 .... xn])
def coNewton(x, a):
    y = f1(x,a)
    num, den, coef, n = [],[],[y[0]],len(x)
    for j in range(1, n):
        for i in range(1, len(y)):
            #por la 1 vez
            i2 = (2*i-1) if j > 1 else i
            dif = (y[i] - y[i-1])/(x[i2] - x[i2-1])
            num.append(dif)
            den.extend([x[i2],x[i2-1]])
        if len(den)>0:
            den.pop(0)
            den.pop(-1)
        if len(num)>0: coef.append(num[0])
        y, x = num, den
        num, den = [], []
    return coef

#polinter: Polinomio interpolador D.D.Newton dado los puntos de Chebyshev.
#Recibe: x a evaluar, los coeficientes de Newton y los puntos de Chebyshev.
#Retorna: El polinomio interpolador evaluado en x.
def polinter(x, coef, pnts):
    Px = coef[0]
    largo, mult = len(pnts), 1
    for i in range(1,largo):
        for j in range(0,i):
            mult = mult*(x-pnts[j])
        Px = Px + coef[i]*mult
        mult = 1
    return Px

#plot1: gráfica 2 funciones en la recta de valores x. k es el valor
#que representa el k-ésimo polinomio de Chebyshev
def plot1(f1, f2, xi, k):
    plt.plot(xi, f1,'r', xi, f2, 'b')
    plt.ylabel('f(x,1) (rojo), P(x) (azul)')
    plt.xlabel('x')
    plt.axis([-1,1,0,1.5])
    plt.title('f(x,1) vs P'+str(k)+'(x)')
    plt.savefig('../Informe/Graficos/pol'+str(k)+'.png', dpi=60)
    plt.hold(False)

#plot2: gráfica el Error entre la función "real" y la interpoladora versus
#el número de puntos de Chebyshev
def plot2(error, n, name):
    plt.plot(n, error)
    plt.ylabel('log_10(En(1))')
    plt.xlabel('n')
```

```
    plt.yscale('log')
    plt.title('log10(En(1)) vs '+name+' puntos de Chebyshev)')
    plt.savefig('../Informe/Graficos/Error-vs-n3.png')
    plt.hold(False)

def plot3(pend, alpha):
    plt.plot(alpha, pend)
    plt.ylabel('Pendiente')
    plt.axis([0,270,-5,0])
    plt.xlabel('Alpha')
    plt.title("Pendiente vs Alpha")
    plt.savefig('../Informe/Graficos/SvsA.png')
    plt.hold(False)


### Actividad ###
def err(a,b):
    return abs(a-b)
def logscale(x):
    return log10(x)

## Datos ##
# Puntos x-espaciados
xi = arange(-1,1,0.0001)
error, pend = [], []
deb = True # Debugg
# Alpha de la función
alpha = arange(-3,3,0.6)
alpha = [10**(i) for i in alpha]
# Alpha = range(1,2)    # Comentar para variar el alpha
# Puntos de Chebyshev
n = range(3,8)
# Nombre plot2 para b) o d)
name = '2:100'

# Cambiar los parámetros de este bucle y des-comentar para generar los gráficos.
if deb:
    for a in alpha:
        for k in n:
            cP = chebPoints(k)              # Puntos de Chebyshev
            coef = coNewton(cP,a)           # Coeficientes D.D.Newton
            Px = polinter(xi, coef, cP)     # Función interpoladora en xi
            fx = f1(xi,a)                   # Función dada en xi
            #plot1(fx, Px, xi,k)
            error.append(max(err(fx,Px)))   # Máx valor error
        #plot2(error, n, name)
        pend.append(polyfit(logscale(error), n, 1)[0])  # Pendiente
        error = []
#   plot3(pend,alpha)
```