Práctica 3: Interpolación

Jose Antonio Lorencio Abril, 3º PCEO

Ejercicios

1.

diferencias_divididas: Calcula la tabla de diferencias divididas de un polinomio interpolador con n+1 nodos usando una función recursiva. Se le pasan como parámetros dos vectores, que son x (los nodos), e y (el valor de la función para cada nodo). n se obtiene por el tamaño de x e y.

polinomio_interpolador_Newton: Obtiene el polinomio interpolador en la forma de Newton a partir de una tabla de diferencias divididas previamente calculada. Lo deja expresado como cadena de caracteres en la forma de Horner

coef_polinomio_interpolador: Igual que el anterior, pero el polinomio queda expresado como un vector de sus coeficientes.

```
x=[1, 2, 3, 4];
y=[1, 7, 4, 2];
n=3;
polyfit(x, y, n)
coef_polinomio_interpolador(diferencias_divididas(x,y),x)
```

Como podemos ver a la salida del programa 'ejercicio1.m', es equivalente utilizar las funciones polyfit(x,y,n) y coef_polinomio_interpolador(diferencias_divididas(x,y),x).

```
pol =

    1.6667  -14.5000   37.8333  -24.0000

ans =

    1.6667  -14.5000   37.8333  -24.0000
```

2.

La tabla de diferencias divididas queda:

```
Numero de nodos de interpolacion= 4
Nodos de interpolacion:
   0.30000   0.37000   0.41000   0.52000
Valores de la funcion en los nodos de interpolacion:
   0.97741   0.96557   0.95766   0.93157
La tabla de diferencias divididas es:
          f[.]           f[..]                f[...]
   0.97741   0.00000   0.00000   0.00000
   0.96557  -0.16914   0.00000   0.00000
   0.95766  -0.19775  -0.26006   0.00000
   0.93157  -0.23718  -0.26288  -0.01279
pol =

  -1.2790e-02  -2.4625e-01   1.6745e-04   9.9987e-01
```

a)

```
%calculamos el polinomio de grado 3
p3=coef_polinomio_interpolador(difdiv,x);

%evaluamos en 0.47
yy=polyval(p3, 0.47);
fx=0.94423;

%calculamos los errores
e_a = abs(fx-yy);
e_r = e_a / fx;

%muestro el resultado
printf('P3(0.47)=%d, con Error Absoluto %d y Error Relativo %d\n', yy, e_a, e_r);
```

Cuya salida es:

```
P3(0.47)=0.944222, con Error Absoluto 8.43566e-06 y Error Relativo 8.9339e-06
```

b) En este caso, a la hora de programar no notamos diferencia entre añadir el punto en su lugar ordenado o al final, pues debemos hacer uso de la función diferencias_divididas en ambos casos, y hará todos los cálculos.

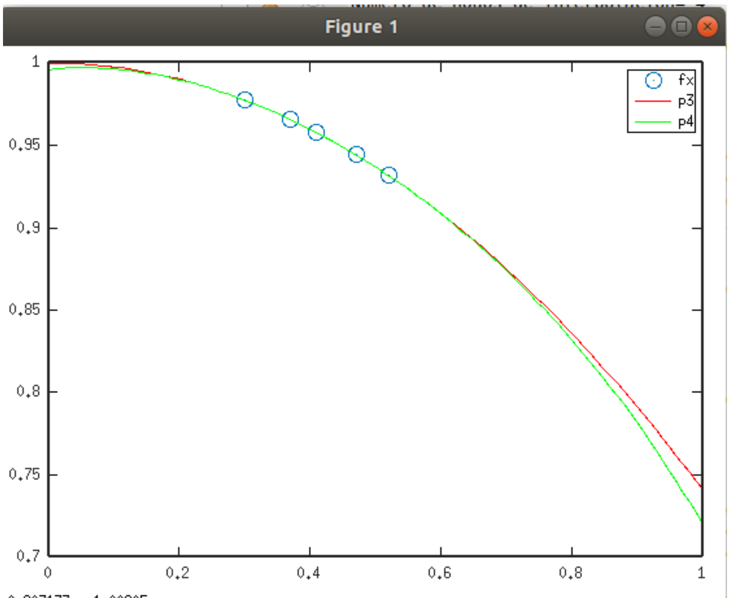
Eso sí, si lo hiciésemos a mano, claramente sería mejor ponerlo al final, pues podríamos reciclar toda la tabla anterior y solo calcular una línea, mientras que si lo pusiéramos en orden, deberíamos desechar varias filas, para recalcularlas. Puede observarse en la siguiente tabla de diferencias divididas que, comparada con la anterior, solo añade la última fila:

```
Numero de nodos de interpolacion= 5
Nodos de interpolacion:
   0.30000   0.37000   0.41000   0.52000   0.47000
Valores de la funcion en los nodos de interpolacion:
   0.97741   0.96557   0.95766   0.93157   0.94423
La tabla de diferencias divididas es:
          f[.]           f[..]                    f[...]
   0.97741   0.00000   0.00000   0.00000   0.00000
   0.96557  -0.16914   0.00000   0.00000   0.00000
   0.95766  -0.19775  -0.26006   0.00000   0.00000
   0.93157  -0.23718  -0.26288  -0.01279   0.00000
   0.94423  -0.25320  -0.26697  -0.04091  -0.16541
pol =

  -0.165405   0.251858  -0.402940   0.040869   0.995953
```

Que ha sido generada con el siguiente código:

```
%añado el nuevo punto y calculo P4
x=[0.3,0.37,0.41,0.52,0.47];
y=[0.97741,0.96557,0.95766,0.93157,0.94423];
difdiv=diferencias_divididas(x,y);
p4=coef_polinomio_interpolador(difdiv,x);

%para dibujar los polinomios
xx=linspace(0,1);
yy=polyval(p3, xx);
zz=polyval(p4,xx);

plot(x,y,'o',xx,yy,'r',xx,zz,'color','green','r');
legend('fx', 'p3', 'p4');
```

Al final del programa, muestro una gráfica en la que se comparan los dos polinomios, que se ve como son muy similares en el intervalo que nos importa, y solo difieren significativamente fuera de este:

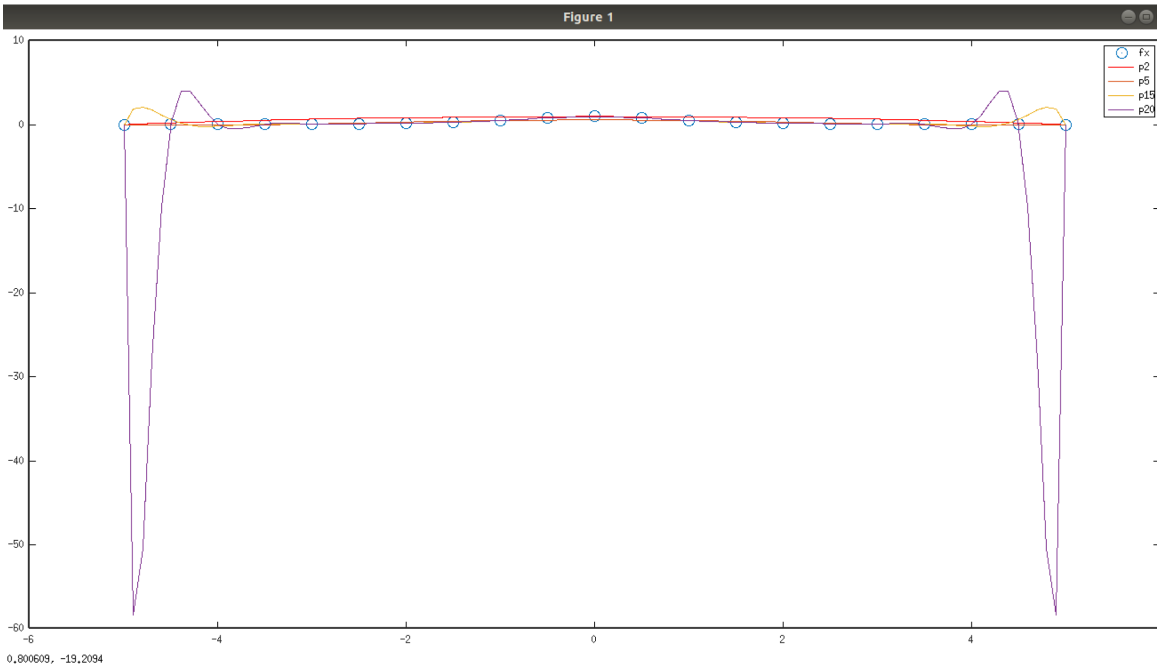


3.

a) En este caso, P_{20} lo muestro logarítmicamente, pues se dispara mucho y si no, no pueden identificarse las diferencias. Esto lo haré más veces en adelante.

```
%divido el intervalo en n+1 puntos equiespaciados
x2=linspace(-5,5,3);
x5=linspace(-5,5,6);
x15=linspace(-5,5,16);
x20=linspace(-5,5,21);

%vectores donde guardaré el valor de la función para los nodos de interpolación
y2=zeros(1,2);
y5=zeros(1,5);
y15=zeros(1,15);
y20=zeros(1,20);

%defino la función
f=@(x)1./(1+x.^2);

y2=f(x2);
y5=f(x5);
y15=f(x15);
y20=f(x20);

%calculo Pn
p2=coef_polinomio_interpolador(diferencias_divididas(x2,y2),x2); p5=coef_polinomio_

%divido el intervalo en 100 puntos equiespaciados y calculo f(x) y Pn(x) en todos e
xx=linspace(-5,5);
z2=polyval(p2, xx);
z5=polyval(p5,xx);
z15=polyval(p15,xx);
z20=polyval(p20,xx);

%muestro la figura
figure(1);

plot(x20,y20,'o',xx,z2,'r',xx,z5,xx,z15,xx,log(z20));
legend('fx', 'p2', 'p5', 'p15', 'p20');
```

Y la gráfica obtenida es la siguiente:



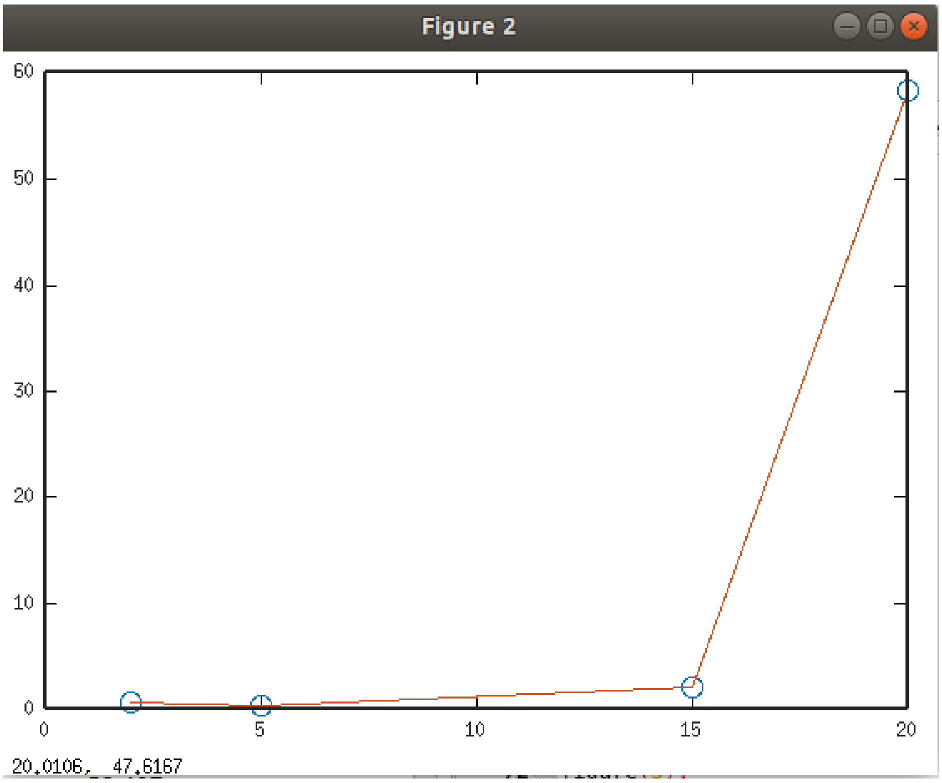
Podemos ver como, cuando crece n, la diferencia se dispara cerca de los extremos del intervalo.

b) En la gráfica podemos observar como el error con P_5 es menor que con P_2, sin embargo, este parece aumentar conforme aumentamos n a partir de este punto.

```
xx2=linspace(-5,5);
fx=f(xx2);

%calculamos los errores absolutos y obtenemos el máximo
zz2=polyval(p2,xx2);
error=abs(fx-zz2);
max2=max(error);

zz5=polyval(p5,xx2);
error=abs(fx-zz5);
max5=max(error);

zz15=polyval(p15,xx2);
error=abs(fx-zz15);
max15=max(error);

zz20=polyval(p20,xx2);
error=abs(fx-zz20);
max20=max(error);

%muestro la evolución del máximo de los errores en una gráfica
n=[2,5,15,20];
er=[max2,max5,max15,max20];
printf('Los máximos son: \n'); %e imprimo los valores
printf('P2: %d, P5: %d, P15: %d, P20: %d',max2,max5,max15,max20);

figure(2);
plot(n,er,'o',n,er);
```

```
Los máximos son:
P2: 0.64597, P5: 0.430325, P15: 2.09672, P20: 58.4067
```

Y nos muestra la gráfica siguiente:

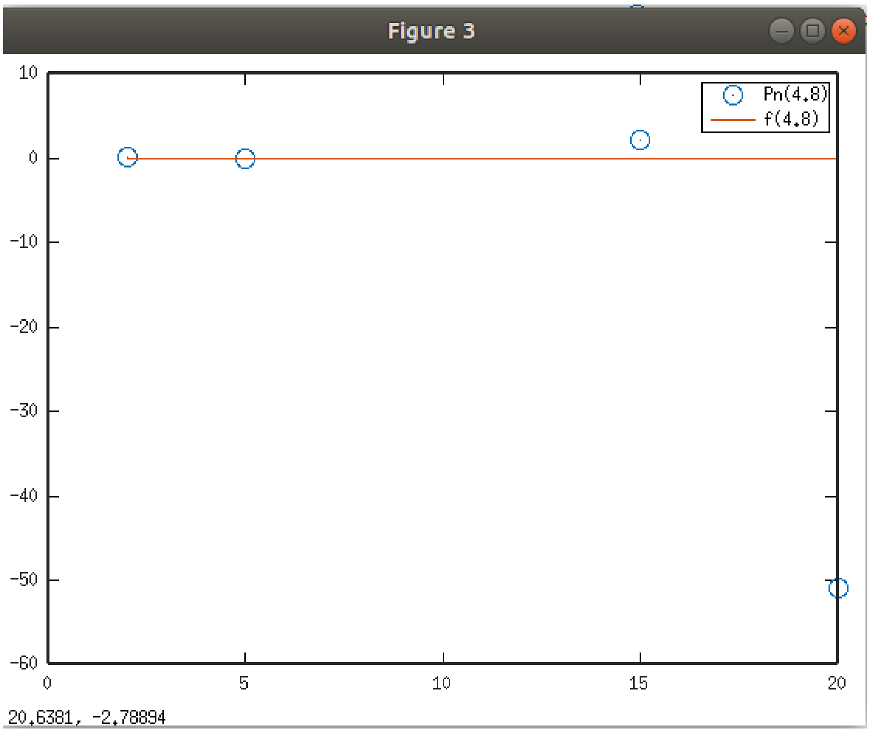


En la que podemos observar como el error parece disminuir en los primeros polinomios, pero después se dispara.

c) Vemos en la Figure 2 que cerca de los extremos, los polinomios se alejan mucho de la función. Vamos a probar con alguno de estos puntos.

```
%elijo un punto que, a ojo, parece que puede servir
posible=4.8;

%vector con los valores de los 4 polinomios en dicho punto
fp=[polyval(p2,posible),polyval(p5,posible),polyval(p15,posible),polyval(p20,posibl
fr=f(posible);
freal=[fr,fr,fr,fr];

%muestro los valores comparados con f(4.8)
figure(3);
plot(n, fp, 'o', n, freal);
legend('Pn(4.8)', 'f(4.8)');
```

En la figura 3, vemos como al aumentar n, el valor arrojado por los polinomios dista mucho del valor de f, por ejemplo, en el punto usado, el 4.8.

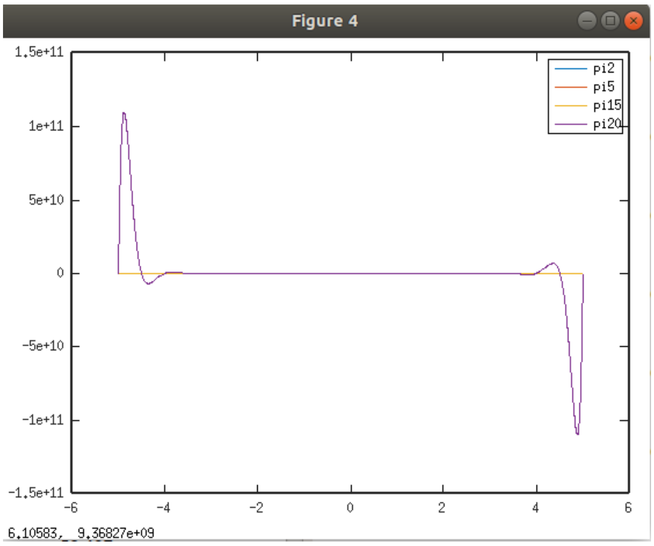


d)

La función pi_n(x,y) calcula π_n para los valores contenidos en x y los nodos de interpolación contenidos en y.

```
%función que calcula pi_n
%x: vector con los puntos en los que queremos calcular pi_n
%y: vector con los nodos de interpolación
function ret = pi_n (x,y)
ret=ones(1,length(x));
cont=1;
for i=x
 for j=y
     ret(cont)=ret(cont)*(i-j);
  end        cont=cont+1;
 end
end

xx=linspace(-5,5);
pi2=pi_n(xx, x2);
pi5=pi_n(xx,x5);
pi15=pi_n(xx,x15);
pi20=pi_n(xx,x20);

%mostramos los 4 polinomios
figure(4);
plot(xx,pi2,xx,pi5,xx,pi15,xx,log(pi20));
legend('pi2', 'pi5', 'pi15', 'pi20');
```

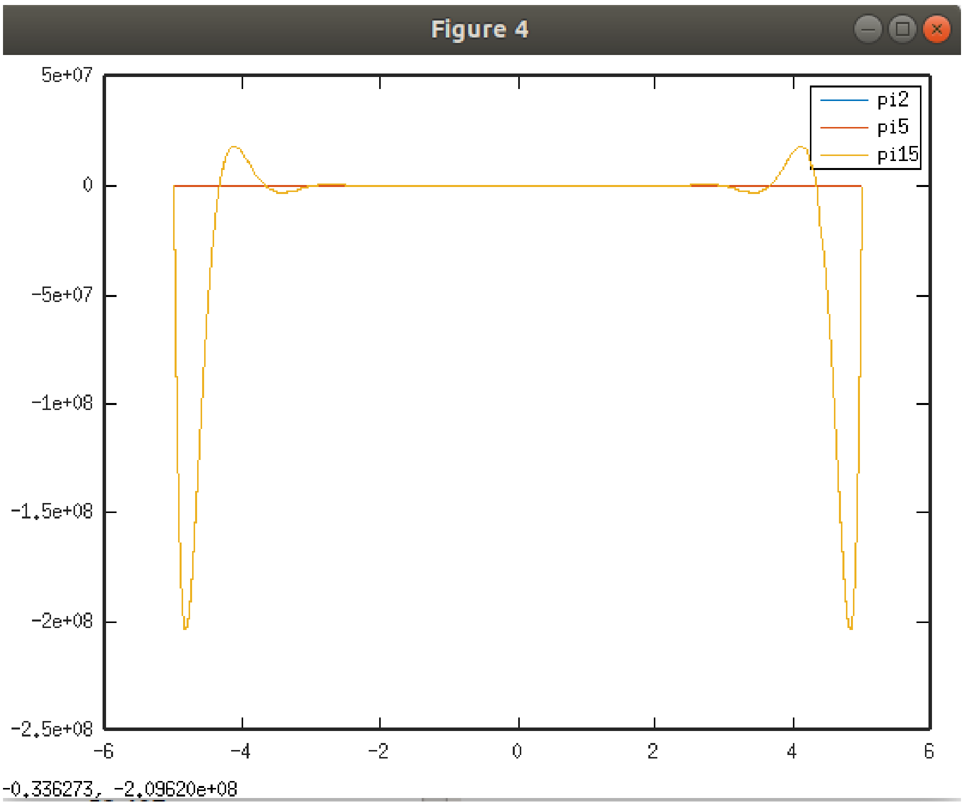
La gráfica que nos muestra el programa:



Donde vemos como π_{20} se aleja muchísimo de los demás, tanto que no podemos compararlos visualmente.

Voy a hacer lo mismo, pero solo mostrando los otros 3 polinomios:

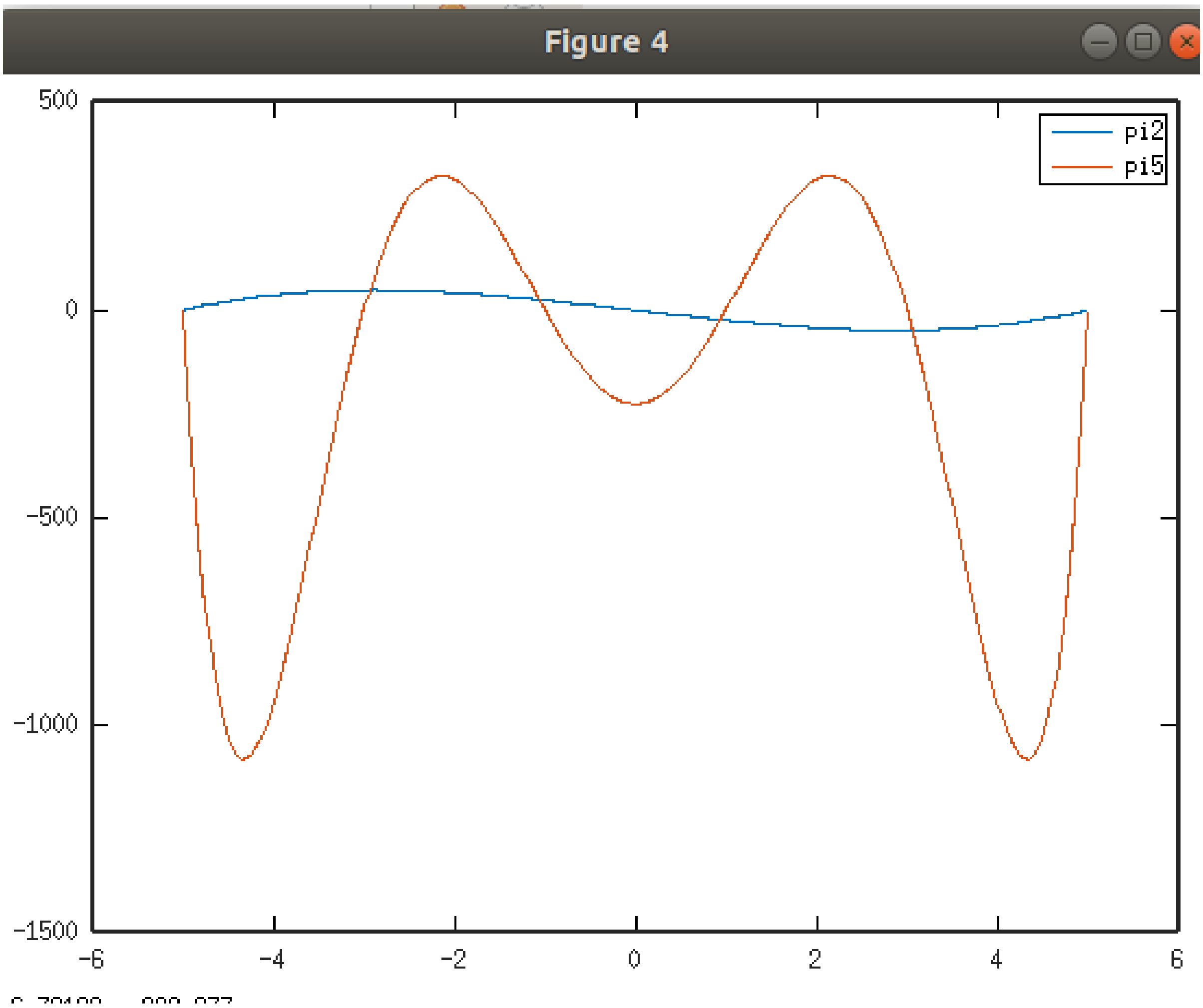
```
plot(xx,pi2,xx,pi5,xx,pi15);
legend('pi2', 'pi5', 'pi15');
```



Ahora podemos observar gran diferencia entre π_{15} y los demás polinomios. Podemos concluir, que al contruir los polinomios de esta forma, cuando aumentamos n, producen grandes oscilaciones en los extremos. Esto es lo que causa los grandes errores en el apartado anterior.

Vamos a dibujar π_2 y π_5:

```
plot(xx,pi2,xx,pi5);
legend('pi2', 'pi5');
```



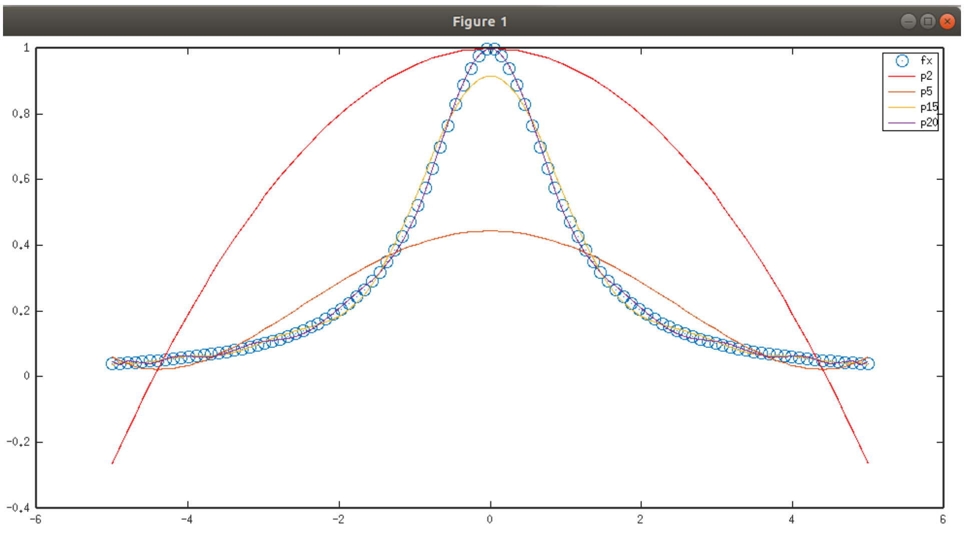
4.

a)

```
x2=zeros(1,3); %los vectores en los que guardaré los nodos
x5=zeros(1,6);
x15=zeros(1,16);
x20=zeros(1,21);

C=@(k,n) 5*cos(((2*k+1)*pi)/(2*(n+1))); %la función de Chebychev
f=@(x)1./(1+x.^2);

for i=[0:2] %calculo los nodos
 x2(i+1)=C(i,2);
end

for i=[0:5]
   x5(i+1)=C(i,5);
end
```

```
for i=[0:15]
 x15(i+1)=C(i,15);
end

for i=[0:20]
  x20(i+1)=C(i,20);
end

y2=f(x2); %calculo el valor de la función en los nodos
y5=f(x5);
y15=f(x15);
y20=f(x20);

%hallo los polinomios de interpolación
p2=coef_polinomio_interpolador(diferencias_divididas(x2,y2),x2);  p5=coef_polinomic

%evaluo los polinomios en 100 puntos del intervalo
xx=linspace(-5,5);
zf=f(xx);
z2=polyval(p2, xx);
z5=polyval(p5,xx);
z15=polyval(p15,xx);
z20=polyval(p20,xx);

%dibujo los polinomios y f(x)
figure(1);
plot(xx,zf,'o',xx,z2,'r',xx,z5,xx,z15,xx,z20);
legend('fx', 'p2', 'p5', 'p15', 'p20');
```

Esto nos muestra la siguiente gráfica:

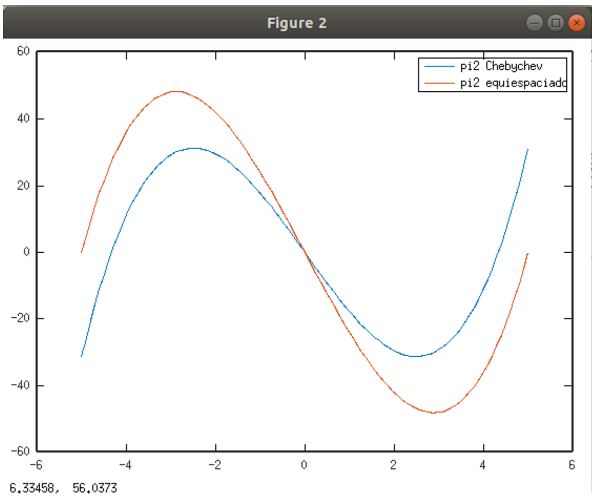


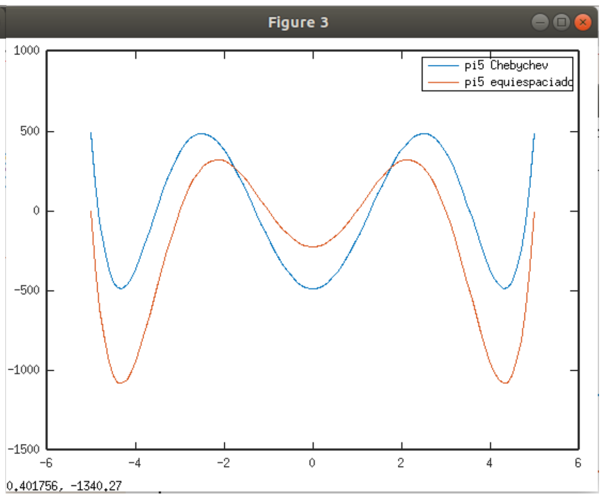
Ahora, al contrario de lo que pasaba antes, vemos como al aumentar n, nuestros polinomios se adecúan cada vez mejor a f(x), haciendo desaparecer el efecto de Runge mediante la utilización de los nodos de Chebychev.

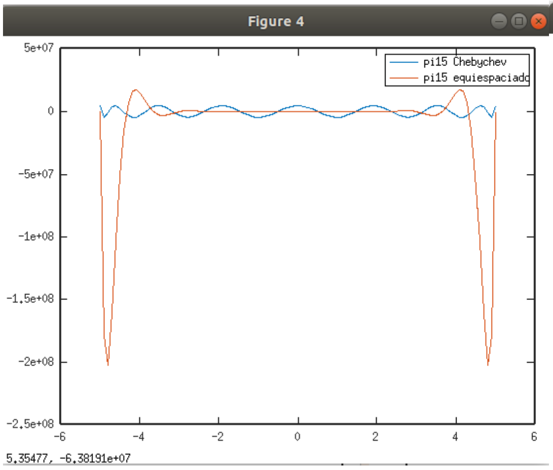
b) Mostrar todos los pi_n a la vez es jeroglífico, por tanto, he optado por mostrarlos por parejas.

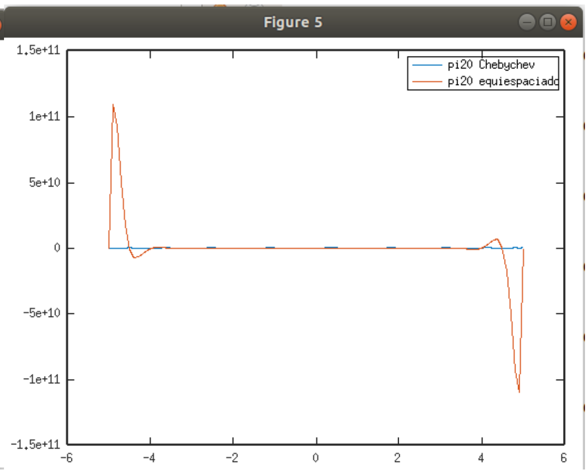
```
%en xn están guardados los nodos de chebychev del polinomio de grado n:
%calculo pi_n para los nodos de Chebychev
xx=linspace(-5,5);
pi2Cheb=pi_n(xx, x2);
pi5Cheb=pi_n(xx,x5);
pi15Cheb=pi_n(xx,x15);
pi20Cheb=pi_n(xx,x20);

%cambio los nodos para que sean equiespaciados
x2=linspace(-5,5,3);
x5=linspace(-5,5,6);
x15=linspace(-5,5,16);
x20=linspace(-5,5,21);

%calculo pi_n para nodos equiespaciados
pi2Eq=pi_n(xx, x2);
pi5Eq=pi_n(xx,x5);
pi15Eq=pi_n(xx,x15);
pi20Eq=pi_n(xx,x20);

%dibujo los pi_2, pi_5, pi_15, pi_20 en parejas
figure(2);
plot(xx,pi2Cheb,xx,pi2Eq); legend('pi2 Chebychev', 'pi2 equiespaciado');
figure(3);
plot(xx,pi5Cheb,xx,pi5Eq); legend('pi5 Chebychev','pi5 equiespaciado');
figure(4);
plot(xx,pi15Cheb,xx,pi15Eq); legend('pi15 Chebychev','pi15 equiespaciado');
figure(5);
plot(xx,pi20Cheb,xx,pi20Eq); legend('pi20 Chebychev', 'pi20 equiespaciado');
```

Y obtenemos las gráficas siguientes:







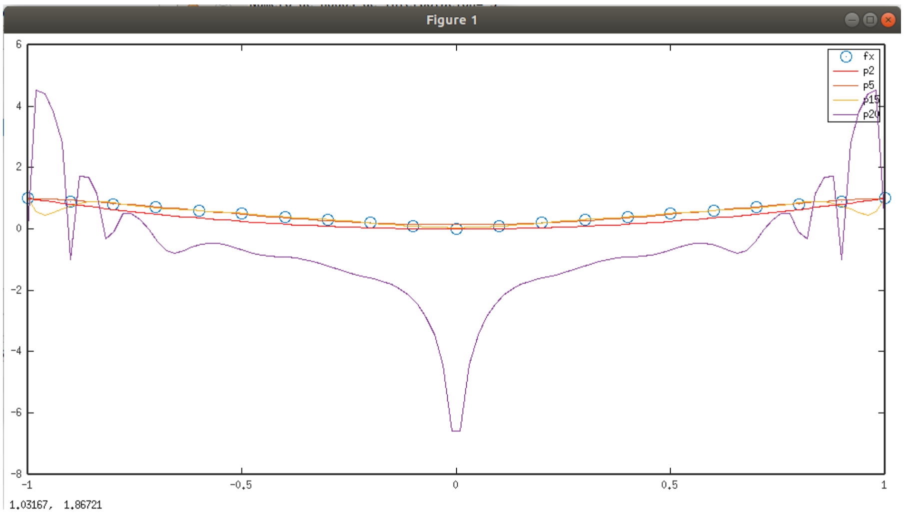


Podemos observar como los polinomios creados con puntos equiespaciados van oscilando mucho más en los extremos que los creados con los nodos de Chebychev, que se mantienen más suaves en todo el intervalo.

5.

Es repetir todo lo anterior, pero cambiando la función utilizada y el intervalo:

```
x2=linspace(-1,1,3);
x5=linspace(-1,1,6);
x15=linspace(-1,1,16);
x20=linspace(-1,1,21);

y2=zeros(1,2); y5=zeros(1,5); y15=zeros(1,15); y20=zeros(1,20);

%definimos la nueva f(x)
f=@(x)abs(x);

y2=f(x2); y5=f(x5); y15=f(x15); y20=f(x20);

%calculamos los polinomios equiespaciados
p2=coef_polinomio_interpolador(diferencias_divididas(x2,y2),x2); p5=coef_polinomio_

%los evaluamos en 100 puntos del intervalo y los dibujamos
xx=linspace(-1,1);
z2=polyval(p2, xx);
z5=polyval(p5,xx);
z15=polyval(p15,xx);
z20=polyval(p20,xx);

figure(1);
plot(x20,y20,'o',xx,z2,'r',xx,z5,xx,z15,xx,log(z20));
legend('fx', 'p2', 'p5', 'p15', 'p20');
```

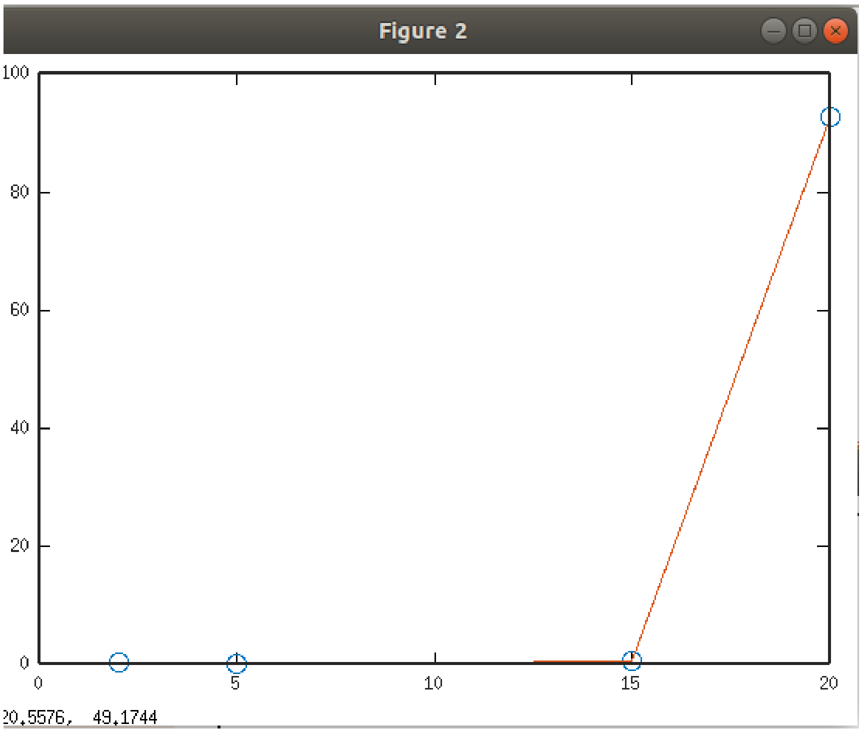


Donde observamos que las aproximaciones van empeorando a partir de cierto n.

Nótese que P_{20} ha sido mostrado en escala logarítmica, es decir, que la diferencia es aun mayor de lo que se aprecia en la figura.

```
xx2=linspace(-1,1);

fx=f(xx2);

%calculamos los errores en 100 puntos y obtenemos el máximo para cada n
zz2=polyval(p2,xx2);
error=abs(fx-zz2);
max2=max(error);

zz5=polyval(p5,xx2);
error=abs(fx-zz5);
max5=max(error);

zz15=polyval(p15,xx2);
error=abs(fx-zz15);
max15=max(error);

zz20=polyval(p20,xx2);
error=abs(fx-zz20);
max20=max(error);

%dibujamos los errores
n=[2,5,15,20];
er=[max2,max5,max15,max20];

printf('Los máximos son: \n'); %y los imprimo
printf('P2: %d, P5: %d, P15: %d, P20: %d',max2,max5,max15,max20);

figure(2);
plot(n,er,'o', n, er);
```

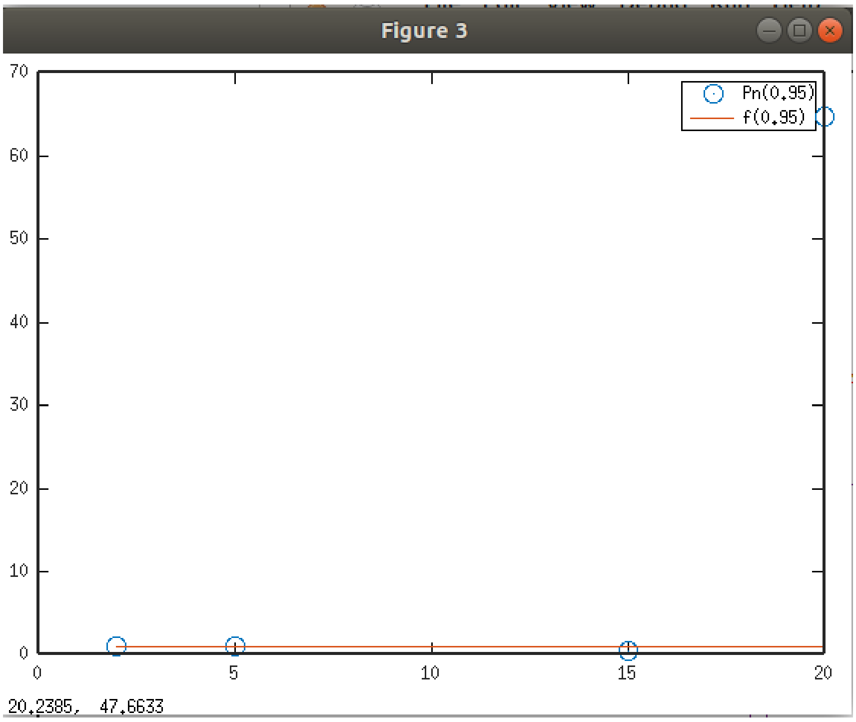
```
Los máximos son:
P2: 0.249974, P5: 0.130678, P15: 0.502852, P20: 92.905
```



Aquí podemos observar como el error aumenta a partir de cierto n, ciertamente se dispara.

Para ver que hay ciertos puntos donde no hay convergencia, he hecho lo mismo que antes, eligiendo esta vez x=4.65.

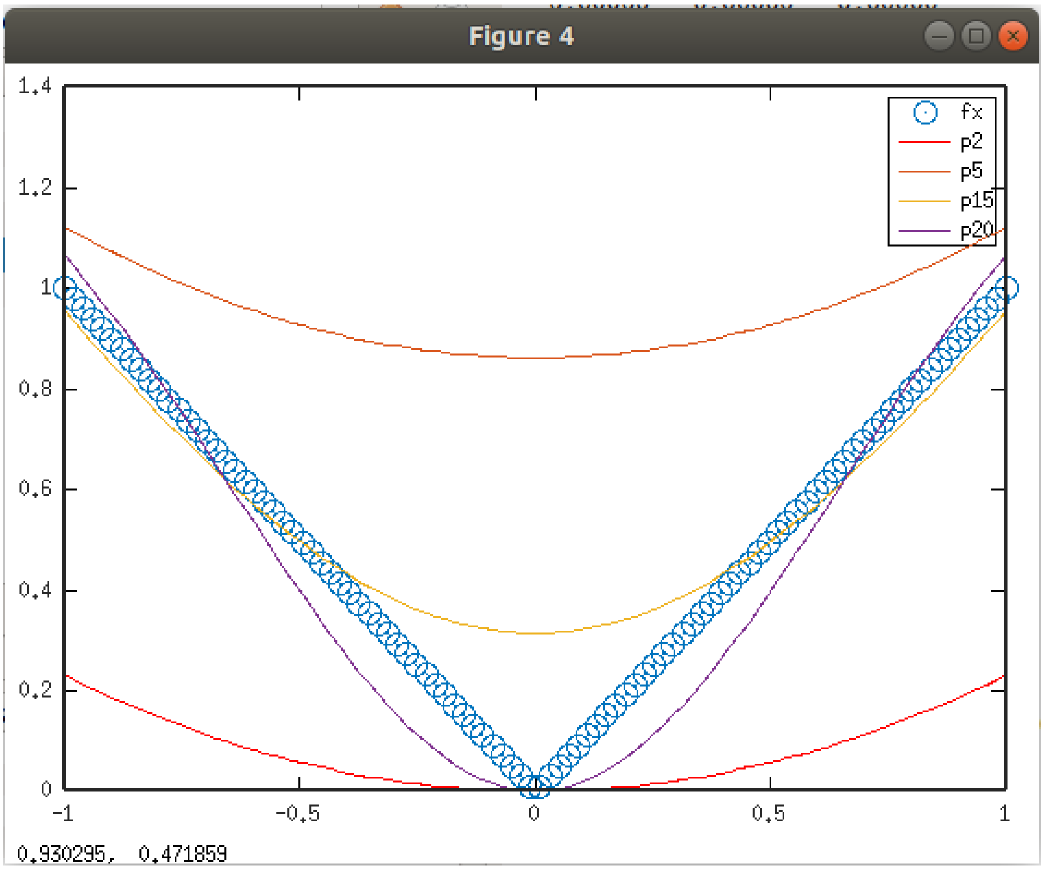
```
posible=0.95;
fp=[polyval(p2, posible), polyval(p5, posible), polyval(p15, posible), polyval(p20, posibl
fr=f(posible);
freal=[fr, fr, fr, fr];

figure(3);
plot(n, fp, 'o', n, freal);
legend('Pn(0.95)', 'f(0.95)');
```



Podemos observar como para n=20, el valor difiere mucho del esperado. Presumimos, así, que esto ocurrirá para mayores n.

```
%chebychev, mismo proceso que antes
x2=zeros(1,3);
x5=zeros(1,6);
x15=zeros(1,16);
x20=zeros(1,21);

C=@(k,n) 5*cos(((2*k+1)*pi)/(2*(n+1)));

for i=[0:2]
        x2(i+1)=C(i,2);
end

for i=[0:5]
        x5(i+1)=C(i,5);
end

for i=[0:15]
        x15(i+1)=C(i,15);
end

for i=[0:20]
        x20(i+1)=C(i,20);
end

y2=f(x2); y5=f(x5); y15=f(x15); y20=f(x20);

%hallamos los polinomios interpoladores con los nodos de Chebychev
```

```
p2=coef_polinomio_interpolador(diferencias_divididas(x2,y2),x2); p5=coef_polinomio_

%calculamos pn(x) en cien puntos equiespaciados del intervalo
zf=f(xx);
z2=polyval(p2, xx);
z5=polyval(p5,xx);
z15=polyval(p15,xx);
z20=polyval(p20,xx);

%lo mostramos en pantalla
figure(4);
plot(xx,zf,'o',xx,z2,'r',xx,z5,xx,z15,xx,z20); legend('fx', 'p2', 'p5', 'p15', 'p20
```



Y vemos como ahora sí que al aumentar n, aproximamos la función de forma razonable.