PURDUE UNIVERSITY · CS 31400
NUMERICAL ANALYSIS

HOMEWORK
Patrick Talley
November 5, 2016

Please answer the following questions in complete sentences in submit the solution on Blackboard November 7, 2016 by 5pm.

Homework 5

Problem 1 (20 points)

1. Chapter 8, Problem 1, Parts (a), (b) (5 points each)
2. (5 points) Use the setup from chapter 8, Problem 1. Suppose we present the data from part (b) as the number of milliseconds from 1900. (This involves multiplying each "year" by `365*24*60*60*1000`.) Evaluate the quadratic Lagrange interpolant directly, and via the Barycentric formulation at 50 uniformly spaced points between (and including) the interpolation end points (e.g. 1900 and 1940 translated into milliseconds). Report on any differences. (Hint: there should be something weird ...)
3. (5 points) **Note, there are much better ways of evaluating the barycentric Lagrange polynomial.** Here is one that is based on a Matlab code by one of the authors who recognized the importance of Barycentric interpolation.

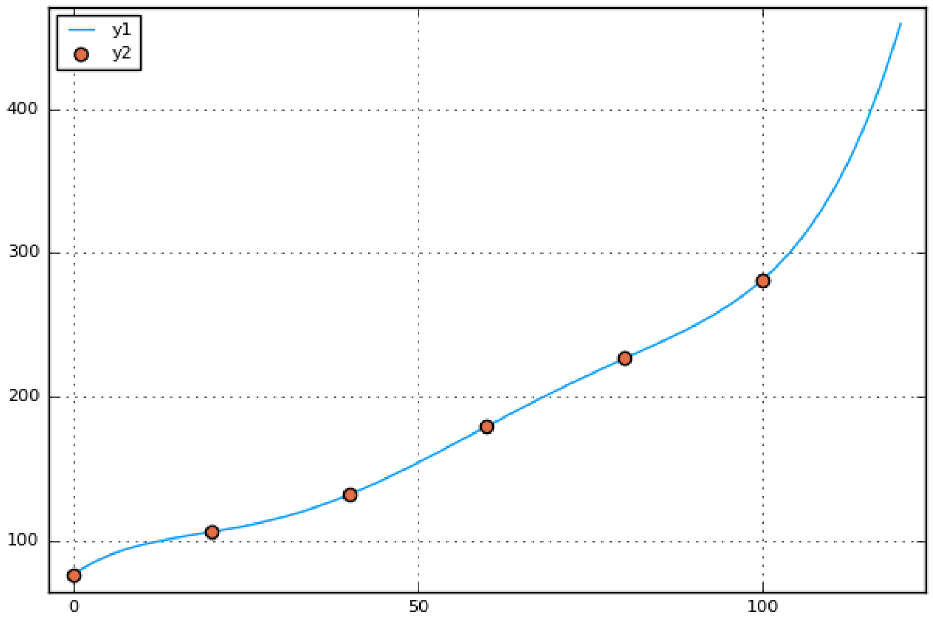
```
function barylag(x,y,xx)
    # direct port of
    # http://www.mathworks.com/matlabcentral/fileexchange/...
    #    4478-barycentric-lagrange-interpolating-polynomials-...
    #    and-lebesgue-constant/content/barylag.m
    # to Julia. Not that better implmentations in Julia are possible.
    M = length(x)
    N = length(xx)
    @assert M == length(y)
    X = repmat(x,1,M)
    W = repmat(1./prod(X-X'+eye(M),1),N,1)
    xdist=repmat(xx,1,M)-repmat(x',N,1)
    fixi,fixj = findnz(xdist.==0)
    H=W./xdist
    p=vec((H*y)./sum(H,2))
    p[fixi] = y[fixj]
    return p
end
```

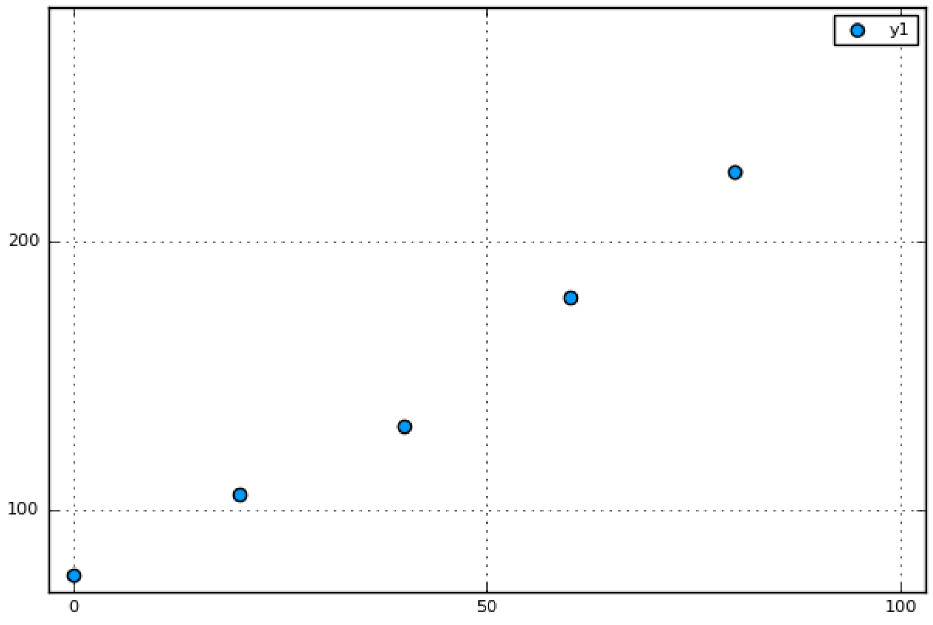
   Use this code to interpolate the data in part (b) instead, using the values of the data in milliseconds. Report on how this code addressed the problem. (That is, you must explain why this code was able to fix the issue you found in part 2.)

Problem 1 Solution

1. (a) Interpolation is not a reasonable way to do prediction because it does not try to get a sense of how the data is trending. This means that it

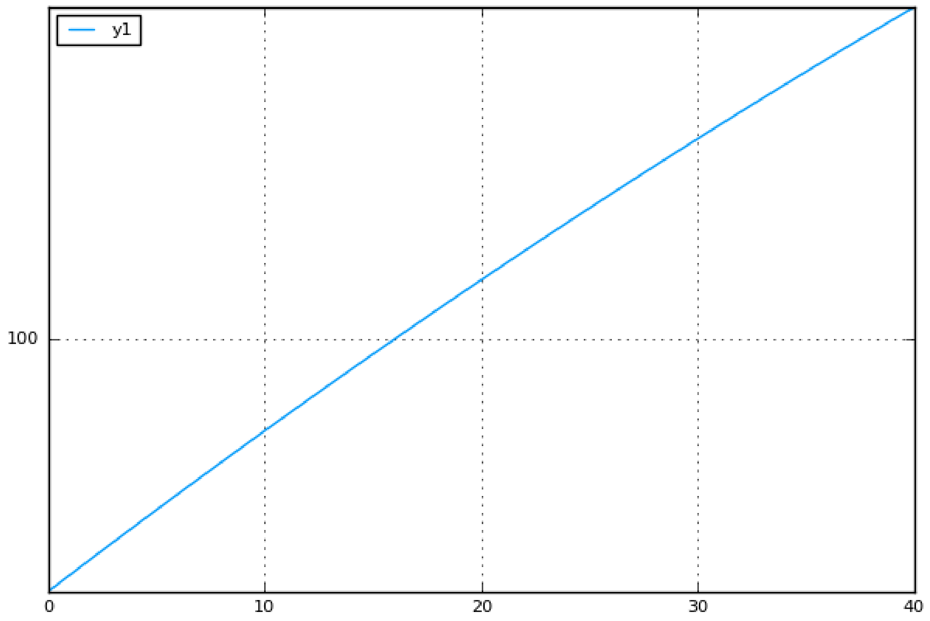
often overfits, and is not representative of the future.



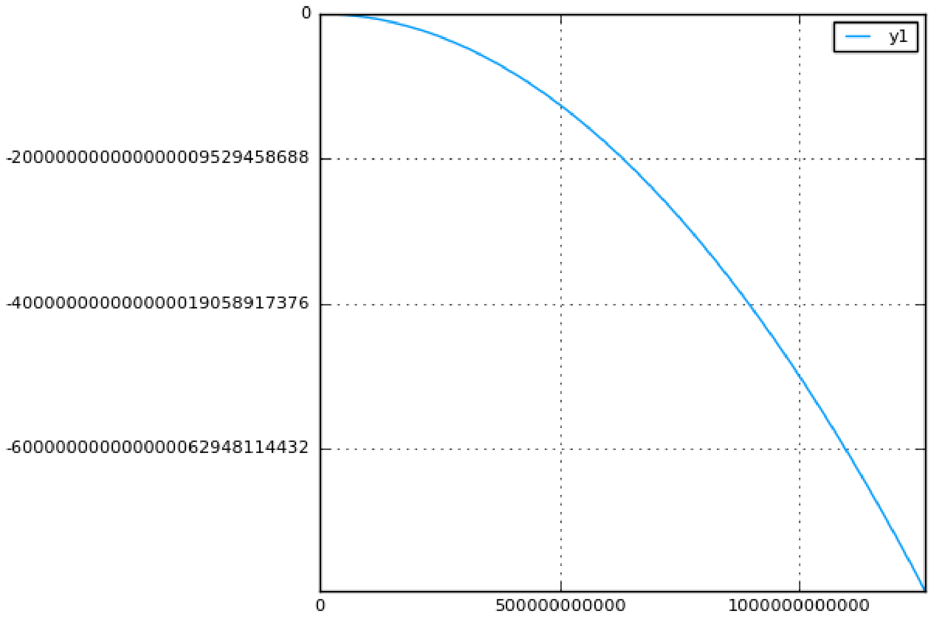


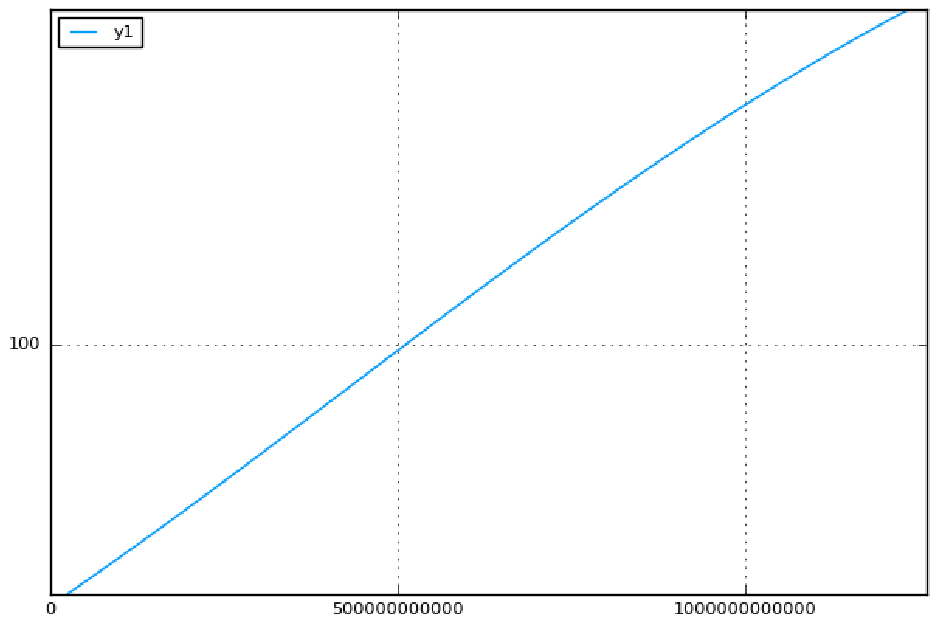
(b)

$$p(x) = 76\frac{(x-20)(x-40)}{(-20)(-40)} + 105.7\frac{x(x-40)}{20(20-40)} + 131.7\frac{x(x-20)}{40(40-20)}$$

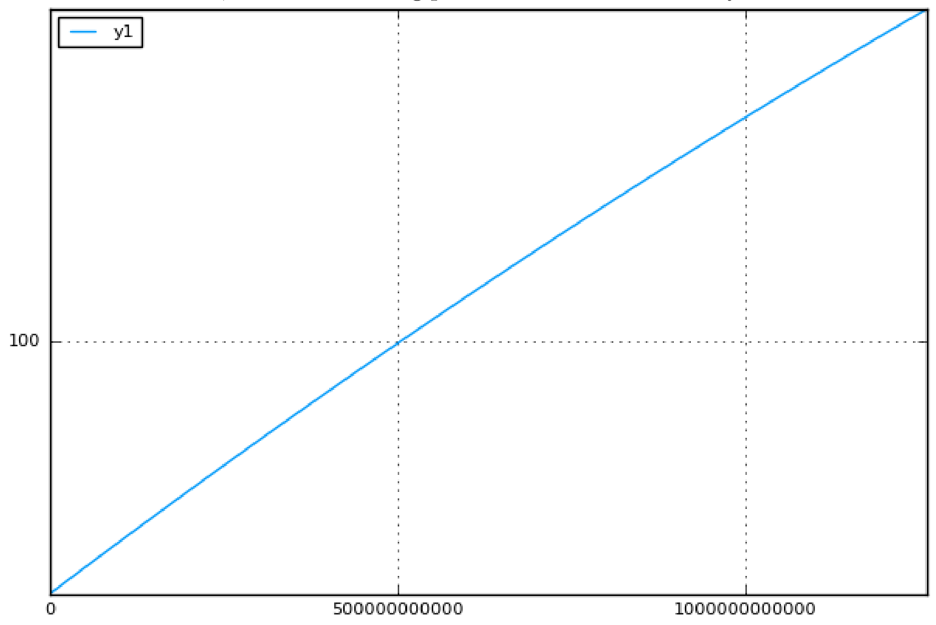


2. The Lagrange interpolant tends towards $-\infty$, and does not interpolate the data correctly at all. The Barycentric interpolant does correctly interpolate the data.





3. The reason this barycentric function is able to correctly interporlate the data, and not run into the same issues that Lagrange form had is that is forces the data to exact at the data points. Lagrange assumes that the calculation will force it to be so, but with floating point error this is not always the case.



Problem1

November 5, 2016

```
In [8]: using Plots
        using ApproxFun
```

1 Part 1

```
In [2]: xi = [0; 20; 40; 60; 80; 100]
        yi = [76.0; 105.7; 131.7; 179.3; 226.5; 281.4]

        scatter(xi,yi)
```

```
[Plots.jl] Initializing backend: pyplot
```

```
In [3]: """
        `lagrange_interp`: lagrange_interp(x, xi, yi)
        y = lagrange_interp(x,xi,yi) uses the data from xi,yi to build the
        Lagrange interpolant and then evaluates that interpolant at the points
        x, returning the interpolated values y.
        """
        function lagrange_interp(x, xi, yi)
          y = zeros(size(x))
          for i=1:length(yi)
            elli = ones(size(x))
            for j=1:length(xi)
              if i==j continue; end
              elli = elli.*(x-xi[j])/(xi[i] - xi[j])
            end
            y = y + yi[i]*elli
          end
          return y
        end
        ## Lagrange form
        xx = linspace(0,120,1000)
        yyl = lagrange_interp(xx,xi,yi);
        plot(xx,yyl); scatter!(xi,yi)
```

$$p(x) = 76\frac{(x-20)(x-40)}{(-20)(-40)} + 105.7\frac{x(x-40)}{20(20-40)} + 131.7\frac{x(x-20)}{40(40-20)}$$

```
In [4]: p1 = x -> 76.0*((x)-(20))*((x)-(40))/((-20)*(-40))
        p2 = x -> 105.7*(x)*((x)-(40))/((20)*(20-40))
        p3 = x -> 131.7*(x)*((x)-(20))/((40)*(40-20))
        f = x -> p1(x)+p2(x)+p3(x)
        n=200
        yl=zeros(n,1)
        tvals = linspace(0,40,n)
        for i=1:n
            yl[i] = f(tvals[i])
        end
        plot(tvals, yl)
```

2 Part 2

Direct computation of Lagrange interpolant

```
In [5]: year_to_ms = 365*24*60*60*1000
        p1 = x -> 76.0*((x*year_to_ms)-(20*year_to_ms))*((x*year_to_ms)-(40*year_to_ms))/((-20*year_to_
        p2 = x -> 105.7*(x*year_to_ms)*((x*year_to_ms)-(40*year_to_ms))/((20*year_to_ms)*(20*year_to_ms
        p3 = x -> 131.7*(x*year_to_ms)*((x*year_to_ms)-(20*year_to_ms))/((40*year_to_ms)*(40*year_to_ms
        f = x -> p1(x)+p2(x)+p3(x)

        n=50
        yl=zeros(n,1)
        tvals = linspace(0,40*year_to_ms,n)
        for i=1:n
            yl[i] = f(tvals[i])
        end
        plot(tvals, yl)


        LoadError: UndefVarError: y2ms not defined
    while loading In[5], in expression starting on line 9
```

Direct Barycentric interpolant

```
In [6]: y2ms = 365*24*60*60*1000
        w0 = 1/((-20*y2ms)*(-40*y2ms))
        w1 = 1/((20*y2ms)*(20*y2ms-40*y2ms))
        w2 = 1/((40*y2ms)*(40*y2ms*20*y2ms))

        p = x -> (w0*yi[1]/(x) + w1*yi[2]/(x-(20*y2ms)) + w2*yi[3]/(x-(40*y2ms))) / (w0/(x) + w1/(x-(20

        n=50
        yb=zeros(n,1)
        tvals = linspace(0,40*y2ms,n)
        for i=1:n
            yb[i] = p(tvals[i])
        end

        plot(tvals, yb)
```

3 Part 3

```
In [7]: function barylag(x,y,xx)
            # direct port of
```

```
    # http://www.mathworks.com/matlabcentral/fileexchange/...
    #    4478-barycentric-lagrange-interpolating-polynomials-...
    #    and-lebesgue-constant/content/barylag.m
    # to Julia. Not that better implmentations in Julia are possible.
    M = length(x)
    N = length(xx)
    @assert M == length(y)
    X = repmat(x,1,M)
    W = repmat(1./prod(X-X'+eye(M),1),N,1)
    xdist=repmat(xx,1,M)-repmat(x',N,1)
    fixi,fixj = findnz(xdist.==0)
    H=W./xdist
    p=vec((H*y)./sum(H,2))
    p[fixi] = y[fixj]
    return p
end
ybl = barylag(xi[1:3]*y2ms, yi[1:3],linspace(0,40*y2ms,50))
plot(tvals, ybl)
```

In [ ]:

Problem 2 (20 points)

1. Chapter 8, Problem 5.
2. Implement a Julia function that computes the following:

```
"""
`chebspace`
===========

A Chebyshev analog of linspace for polynomial interpolation

* `chebspace(a,b)` generates an array of 100 Chebyshev points
between `a` and `b`
* `chebspace(a,b,n)` generates an array of `n` points between a, b
and for `n=1`, this returns b.
function chebspace(a,b,n)
# fill this in!
end
function chebspace(a,b)
    return chebspace(a,b,100)
end
```

(Hint, see equation 8.15)

Problem 2 Solution

1. $l(x) = -\frac{x-b}{a-b} + \frac{x-a}{b-a}$
2.

Problem 2

November 5, 2016

$$l(x) = -\frac{x-b}{a-b} + \frac{x-a}{b-a}$$

```
In [1]: """
        `chebspace`
        ===========

        A Chebyshev analog of linspace for polynomial interpolation

        * `chebspace(a,b)` generates an array of 100 Chebyshev points
        between `a` and `b`
        * `chebspace(a,b,n)` generates an array of `n` points between a, b
        and for `n=1`, this returns b.
        function chebspace(a,b,n)
        # fill this in!
        end
        function chebspace(a,b)
            return chebspace(a,b,100)
        end
        """
        function chebspace(a,b,n=100)
            x = zeros(n,1)
            if n==1
                return b
            else
                for i=1:n
                    x[i] = (a+b)/2 + (b-a)/2 * cos((2*i-1)/(2*n)*pi)
                end
                return x
            end
        end

Out[1]: chebspace (generic function with 2 methods)

In [2]: chebspace(-5,5)

Out[2]: 100x1 Array{Float64,2}:
          4.99938
          4.99445
          4.98459
          4.9698
          4.95012
          4.92555
          4.89611
```

```
  4.86185
  4.82279
  4.77897
  4.73043
  4.67722
  4.6194
  ⋮
 -4.67722
 -4.73043
 -4.77897
 -4.82279
 -4.86185
 -4.89611
 -4.92555
 -4.95012
 -4.9698
 -4.98459
 -4.99445
 -4.99938
```

```
In [ ]:
```

Problem 3 (10 points)

Write a paragraph or two (100-200 words) about what you learned about polynomials and polynomial interpolation. You should also think about answering the following questions:

1. Do you think you'd ever use polynomial interpolation?
2. If so, how?
3. If not, why not?

Be honest, but remember to be thoughtful.

Problem 3 Solution

I do not think I will use polynomial interpolation too often. Most of the time it seems that approximation is more useful for visualization of data sets. The application of in typography is neat, but I do not think that I will ever be designing fonts. I suppose the main limitation I see with interpolation is that data sets are often chaotic, and fitting to everypoint does not give as good of a picture to an approximately correct graph. I would not say I will never use interpolation however, as there are often times when I need to plot a smooth function, and for this application interpolation makes sense.

Problem 4 (10 points)

1. Download and install ApproxFun. Then type `using ApproxFun` to get it added to your Julia instance. Also type `using Plots` to get the plotting interaface.

2. Run

   ```
   f = Fun(x->sign(x))
   plot(f)
   ```

   How does this relate to the Weistrauss approximation theorem?

3. Try using ApproxFun to integrate the most complicated function (in one dimension) that you can think of! Can you stump ApproxFun?

4. One of the advantages of Chebfun is we can get computer representations of functions (as polynomials) that have no analytic form what-so-ever. Consider the following code which builds a polynomial approximation of

   $$\rho(t) = \max |\lambda_i \left(t \begin{bmatrix} 1 & 2 & 0 \\ 0 & 2 & 1 \\ 1 & 0 & 2 \end{bmatrix} + (1-t) \begin{bmatrix} 1 & 1 & 0 \\ 1 & -1 & 1 \\ -1 & 1 & 1 \end{bmatrix} \right) |$$

   here λ_i is just the ith eigenvalue. So this is looking at the largest magnitude eigenvalue. (Example from Chebfun: http://www.chebfun.org/examples/approx/NoisyNonsmooth.html)

   ```
   A = [1 2 0; 0 2 1; 1 0 2.0]
   B = [1 1 0; 1 -1 1; -1 1 1.0]
   f = Fun(t -> maximum(abs(eigvals(t*A + (1-t)*B))),[0,1])
   plot(f)
   ```

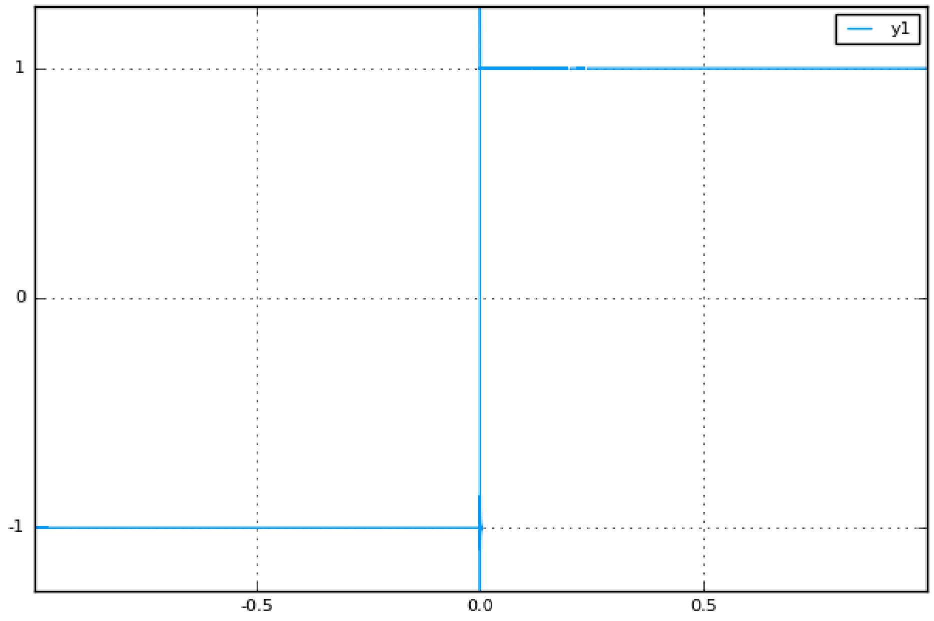
   Run this code and show your result!

5. (Fun bonus question, worth no points.) This requires Matlab. Download and install the Matlab package `chebfun` Run `chebsnake('equi')` (Remember this when you are tempted to use equally spaced points for polynomial interpolation!)

Problem 4 Solution

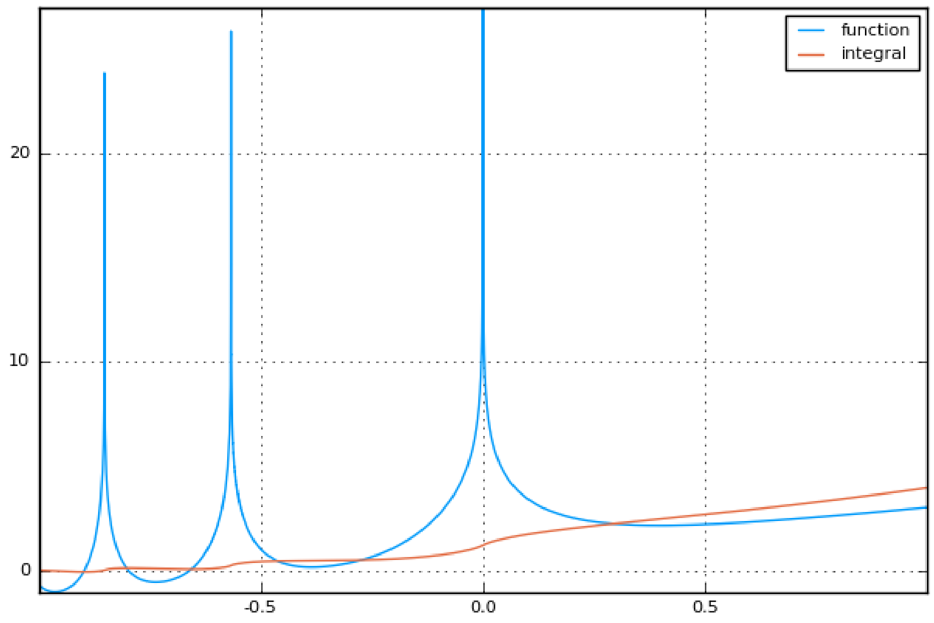
1.

2. The theorem states that any continuous function can be approximated as closely as desired by a polynomial function. The sign function's plot reveals that it is challenging to have the maximum error approach 0, around the point $x = 0$.

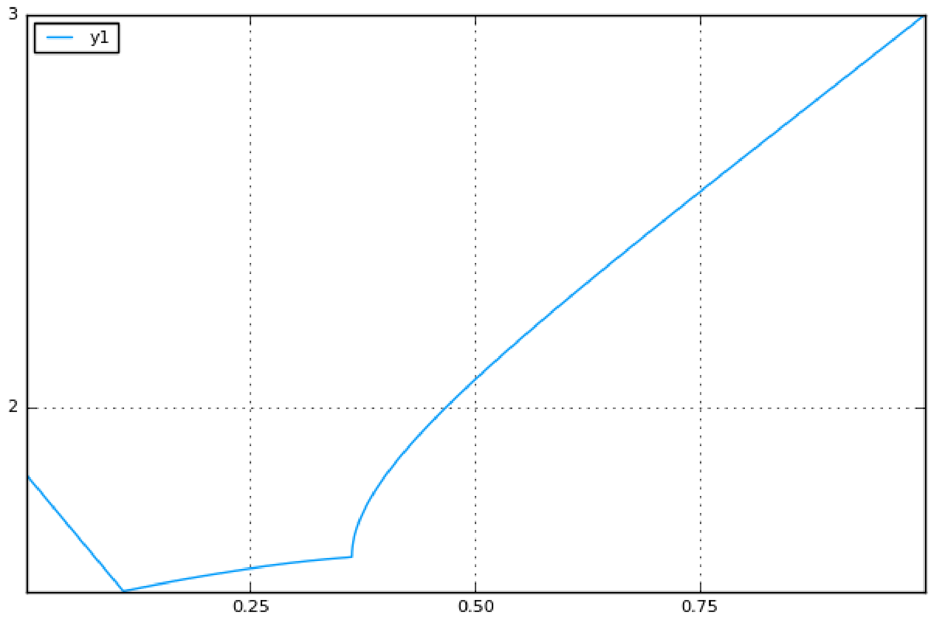


3.

$$f(x) = sinh^{-1}(csc^2(\frac{\pi * x}{e^x})) + 2x$$



4.



5.

Problem4

November 5, 2016

```
In [2]: using ApproxFun
        using Plots

In [3]: f = Fun(x->sign(x))
        plot(f)
        savefig("problem4part2.png")

WARNING: Maximum number of coefficients 1048577 reached

[Plots.jl] Initializing backend: pyplot

In [4]: f = Fun(x->asinh(csc(pi*x/(e^x))^2)+2x)
        g = cumsum(f)
        plot(f, label="function")
        plot!(g, label="integral")
        savefig("problem4part3.png")

WARNING: Maximum number of coefficients 1048577 reached

In [5]: A = [1 2 0; 0 2 1; 1 0 2.0]
        B = [1 1 0; 1 -1 1; -1 1 1.0]
        f = Fun(t -> maximum(abs(eigvals(t*A + (1-t)*B))),[0,1])
        plot(f)
        savefig("problem4part4.png")

WARNING: Maximum number of coefficients 1048577 reached

In [ ]:
```

Problem 5 (40 points)

1. (10 points) Chapter 9, Exercise 1,
2. (10 points) Chapter 9, Exercise 2.
3. (10 points) Chapter 9, Exercise 3.
4. (10 points) Chapter 9, Exercise 5.

Problem 5 Solution

1. The lowest error was observed at $h = 10^{-4}$. One reason for this is that the h term in the denominator is squared, and as observed in central and forward difference, the optimal value is around 10^{-8} digits, so that would correspond to $h \approx 10^{-4}$
2. It is obvious to see how much more quickly the error decreases in Richardson's compared to the previous approximation. For $h = 0.2$ the erro was the same as $h = 10^{-5}$. And for 2 steps of Richardson, the error was much smaller than the best of the previous analysis.
3. error is $1.14908e - 14$
4. $$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(\eta)$$
   $$f(x+2h) = f(x) + 2hf'(x) + \frac{4h^2}{2}f''(x) + \frac{8h^3}{6}f'''(\xi)$$

   Multiply the top equation by 4, and the second by -1, then add the two.

   $$4[f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(\eta)]$$
   $$-[f(x+2h) = f(x) + 2hf'(x) + \frac{4h^2}{2}f''(x) + \frac{8h^3}{6}f'''(\xi)]$$

   $$4f(x+h) - f(x+2h) = 3f(x) + 2hf'(x) + 0 + \frac{2h^3}{3}(f'''(\eta) - f'''(\xi))$$

   Rearrange the terms to solve for $f'(x)$

   $$f'(x) = \frac{1}{2h}[-3f(x) + 4f(x+h) - f(x+2h)] + \frac{h^2}{3}(f'''(\eta) - f'''(\xi))$$

   Therefore, the error term is $O(h^2)$

Problem5

November 5, 2016

```
In [3]: mysecondorder = (x,h) -> (f(x+h) - 2.0*f(x) + f(x-h))/(h^2.0)
        ddf = x -> -1.0*sin(x)
        f = x -> sin(x)

Out[3]: (anonymous function)

In [4]: x = pi/6.0
        for i=1:16
            h=10.0^(-i)
            addf = mysecondorder(x,h)
            err = abs(addf - ddf(x))
            @printf("%.8e  %.16f  %.5e\n", h, addf, err)
        end

1.00000000e-01  -0.4995834721974234  4.16528e-04
1.00000000e-02  -0.4999958333473664  4.16665e-06
1.00000000e-03  -0.4999999583255033  4.16745e-08
1.00000000e-04  -0.4999999969612645  3.03874e-09
1.00000000e-05  -0.5000005964816977  5.96482e-07
1.00000000e-06  -0.4999334279887080  6.65720e-05
1.00000000e-07  -0.4940492459581947  5.95075e-03
1.00000000e-08  -1.1102230246251563  6.10223e-01
1.00000000e-09  55.5111512312578199  5.60112e+01
1.00000000e-10  0.0000000000000000  5.00000e-01
1.00000000e-11  0.0000000000000000  5.00000e-01
1.00000000e-12  0.0000000000000000  5.00000e-01
1.00000000e-13  5551115123.1257820129394531  5.55112e+09
1.00000000e-14  -555111512312.5782470703125000  5.55112e+11
1.00000000e-15  0.0000000000000000  5.00000e-01
1.00000000e-16  -5551115123125783.0000000000000000  5.55112e+15

In [5]: @show mysecondorder(x,0.2)
        @show mysecondorder(x,0.1)
        @show mysecondorder(x,0.05)
        ;

mysecondorder(x,0.2) = -0.49833555396895646
mysecondorder(x,0.1) = -0.4995834721974234
mysecondorder(x,0.05) = -0.4998958420134868

In [7]: h0=0.2

        h1=h0/2.0
        rho0 = mysecondorder(x,h0)
        rho1 = mysecondorder(x,h1)
        addf0 = (4/3)*rho1 - (1/3)*rho0
        err = abs(addf0 - ddf(x))
        @printf("%.8e  %.16f  %.5e\n", h0, addf0, err)
        h0=h1

        h1=h0/2.0
        rho0 = mysecondorder(x,h0)
        rho1 = mysecondorder(x,h1)
        addf1 = (4/3)*rho1 - (1/3)*rho0
        err = abs(addf1 - ddf(x))
        @printf("%.8e  %.16f  %.5e\n", h0, addf1, err)
        h0=h1

        addf2 = (16/15)*addf1 - (1/15)*addf0
        err = abs(addf2 - ddf(x))
        @printf("Combination     %.16f  %.5e\n", addf2, err)
```

```
2.00000000e-01  -0.4999994449402457  5.55060e-07
1.00000000e-01  -0.4999999652855079  3.47145e-08
Combination     -0.4999999999751921  2.48078e-11
```

```
In [8]: using ApproxFun
```

```
In [9]: x=pi/6.0
        f = Fun(x->sin(x))
        aprox = f''(x)
        err = abs(aprox - ddf(x))
        @printf("%.16f  %.5e\n", aprox, err)
```

```
-0.4999999999999885  1.14908e-14
```

```
In [ ]:
```