HW 3 (CSE 5361 Painter) - Drew Ripberger

The solutions are written in Julia.

Code

```
# Numerical Methods Spring 2023
# Professor Nick Painter
#
# Author: Drew Ripberger

using Printf


# DividedDifference algorithm to find a_0, ... , a_n coefficients of the newton polynomial
function DividedDifference(X, Y, n)
    a = zeros(n, n)
    a[:,1] = Y[1:n]
    for i = 2:n
        for j = 1:(n-i+1)
            a[j, i] = (a[j+1,i-1]-a[j,i-1])/(X[i+j-1] - X[j])
        end
    end
    a[1,:]
end

# Helper function to evaluate a newton polynomial of the form
# p(pt) = a[0] + a[1]*(pt-X[0]) + a[2]*(pt-X[0])*(pt-X[1]) ...
function evaluate_newton(X, pt, a)
    total = 0
    n = length(a)
    temp_prod = 1
    for i = 1:n
        total += temp_prod * a[i]
        temp_prod *= pt-X[i]
    end
    total
end

# Used for Problem 4.1.15
X = [-2, -1, 0, 1, 2, 3]
Y = [1, 4, 11, 16, 13, -4]
println(DividedDifference(X, Y, 6))

# Computer Problem 4.1.3
```

```julia
# Since our interval is conveniently [0, 2] lets use Chebychev points
# by shifting the original domain of [-1, 1] right one

# points for the degree 10 polynomial
n = 11
X = collect(map(x -> cos(pi*((2*x + 1)/(2*n + 1))) + 1, 1:n))
Y = collect(map(x -> exp(x), X))

# interpolate polynomial using Chebychev points
p_interp = DividedDifference(X, Y, 11)


# points for comparison
n = 100
X = collect(map(x -> cos(pi*((2*x + 1)/(2*n + 1))) + 1, 1:n))
Y = collect(map(x -> exp(x), X))

p_points = collect(map(pt -> evaluate_newton(X, pt, p_interp), X))

@printf("e^x: exp(x) - p(x)\n")
for p in zip(X, Y, p_points)
    @printf("e^%.6f: %.6f - %.6f\n", p[1], p[2], p[3])
end

# Computer Problem 4.1.9

X = [1.4, 1.25]
Y = [3.7, 3.9]

# Interpolate data as a function of y

p_interp = DividedDifference(Y, X, 2)
root = evaluate_newton(Y, 0, p_interp)
@printf("root found via interpolation: x=%f\n", root)
```

Output

```
[1.0, 3.0, 2.0, -1.0, 0.0, 0.0]
e^x: exp(x) - p(x)
e^1.998901: 7.380939 - 6.801963
e^1.996948: 7.366538 - 6.789576
e^1.994021: 7.345008 - 6.771052
e^1.990123: 7.316430 - 6.746459
e^1.985257: 7.280917 - 6.715885
e^1.979428: 7.238604 - 6.679442
e^1.972643: 7.189653 - 6.637259
e^1.964907: 7.134250 - 6.589488
```

```
e^1.956229: 7.072603 - 6.536296
e^1.946616: 7.004940 - 6.477869
e^1.936078: 6.931511 - 6.414410
e^1.924625: 6.852581 - 6.346135
e^1.912270: 6.768433 - 6.273275
e^1.899022: 6.679361 - 6.196070
e^1.884897: 6.585674 - 6.114773
e^1.869906: 6.487689 - 6.029645
e^1.854066: 6.385732 - 5.940955
e^1.837391: 6.280134 - 5.848975
e^1.819898: 6.171232 - 5.753985
e^1.801604: 6.059361 - 5.656263
e^1.782527: 5.944861 - 5.556093
e^1.762685: 5.828066 - 5.453756
e^1.742098: 5.709309 - 5.349529
e^1.720786: 5.588919 - 5.243691
e^1.698769: 5.467215 - 5.136513
e^1.676070: 5.344512 - 5.028262
e^1.652710: 5.221112 - 4.919197
e^1.628713: 5.097310 - 4.809572
e^1.604101: 4.973387 - 4.699629
e^1.578899: 4.849613 - 4.589604
e^1.553131: 4.726246 - 4.479722
e^1.526823: 4.603528 - 4.370197
e^1.500000: 4.481689 - 4.261233
e^1.472689: 4.360944 - 4.153021
e^1.444915: 4.241492 - 4.045743
e^1.416707: 4.123520 - 3.939566
e^1.388092: 4.007196 - 3.834648
e^1.359097: 3.892678 - 3.731134
e^1.329752: 3.780106 - 3.629156
e^1.300085: 3.669607 - 3.528835
e^1.270124: 3.561294 - 3.430281
e^1.239899: 3.455266 - 3.333592
e^1.209440: 3.351608 - 3.238855
e^1.178777: 3.250395 - 3.146145
e^1.147938: 3.151688 - 3.055530
e^1.116955: 3.055537 - 2.967064
e^1.085858: 2.961981 - 2.880795
e^1.054677: 2.871048 - 2.796760
e^1.023443: 2.782758 - 2.714987
e^0.992185: 2.697122 - 2.635499
e^0.960935: 2.614141 - 2.558308
e^0.929724: 2.533809 - 2.483420
e^0.898581: 2.456115 - 2.410836
e^0.867537: 2.381039 - 2.340550
e^0.836623: 2.308557 - 2.272550
e^0.805868: 2.238638 - 2.206819
e^0.775303: 2.171249 - 2.143337
e^0.744957: 2.106351 - 2.082079
```

```
e^0.714861: 2.043902 - 2.023016
e^0.685043: 1.983857 - 1.966116
e^0.655533: 1.926169 - 1.911345
e^0.626360: 1.870788 - 1.858665
e^0.597551: 1.817663 - 1.808038
e^0.569136: 1.766740 - 1.759423
e^0.541142: 1.717968 - 1.712779
e^0.513596: 1.671291 - 1.668062
e^0.486526: 1.626655 - 1.625229
e^0.459957: 1.584006 - 1.584235
e^0.433916: 1.543289 - 1.545036
e^0.408428: 1.504451 - 1.507588
e^0.383518: 1.467438 - 1.471847
e^0.359210: 1.432198 - 1.437768
e^0.335529: 1.398679 - 1.405309
e^0.312496: 1.366833 - 1.374427
e^0.290136: 1.336609 - 1.345080
e^0.268469: 1.307960 - 1.317228
e^0.247516: 1.280840 - 1.290831
e^0.227299: 1.255206 - 1.265850
e^0.207838: 1.231013 - 1.242249
e^0.189150: 1.208222 - 1.219991
e^0.171254: 1.186792 - 1.199043
e^0.154168: 1.166687 - 1.179370
e^0.137908: 1.147870 - 1.160942
e^0.122491: 1.130309 - 1.143729
e^0.107932: 1.113971 - 1.127702
e^0.094243: 1.098827 - 1.112835
e^0.081440: 1.084848 - 1.099103
e^0.069535: 1.072009 - 1.086482
e^0.058538: 1.060286 - 1.074951
e^0.048462: 1.049655 - 1.064489
e^0.039315: 1.040098 - 1.055080
e^0.031107: 1.031596 - 1.046705
e^0.023845: 1.024132 - 1.039350
e^0.017537: 1.017692 - 1.033002
e^0.012190: 1.012264 - 1.027650
e^0.007807: 1.007838 - 1.023285
e^0.004394: 1.004404 - 1.019897
e^0.001954: 1.001956 - 1.017482
e^0.000489: 1.000489 - 1.016034
e^0.000000: 1.000000 - 1.015552
root found via interpolation: x=4.175000
```