NE583 Test 2

J.R. Powers-Luhn

November 20, 2018

1 Problem 1

2 Problem 2

2.1 Normalization factor

$$
\begin{aligned}
\int_6^7 \psi(E)\,\mathrm{d}E &= \int_6^7 \frac{1}{E}\,\mathrm{d}E \\
&= \log 7 - \log 6 \\
f &= 0.154151
\end{aligned}
$$

2.2 Alpha

$$
\alpha = \frac{(A-1)^2}{(A+1)^2} = 1
$$

2.3 Scattering Cross section

$$
\begin{aligned}
\sigma_s^{gg} &= \int_6^7 \mathrm{d}E' \int_6^7 \mathrm{d}E\,\frac{\sigma}{(1-\alpha)E}\frac{\psi(E)}{f} \\
&= \frac{\sigma}{f}\int_6^7 \mathrm{d}E' \int_6^7 \frac{\mathrm{d}E}{E^2} \\
&= \frac{\sigma}{f}\int_6^7 \mathrm{d}E' \left(\frac{-1}{7} - \frac{-1}{6}\right) \\
&= \frac{\sigma}{f}\left(\frac{1}{6} - \frac{1}{7}\right)(7-6) \\
&= \frac{20\,\mathrm{b}}{0.154151}\left(\frac{1}{6} - \frac{1}{7}\right)(7-6) \\
&= 3.089\,\mathrm{b}
\end{aligned}
$$

3 Problem 3

[11pt]article

[T1]fontenc mathpazo

graphicx caption nolabel labelformat=nolabel

adjustbox xcolor enumerate geometry amsmath amssymb textcomp upquote eurosym [mathletters]ucs [utf8x]inputenc fancyvrb grffile hyperref longtable booktabs [inline]enumitem [normalem]ulem

urlcolorrgb0,.145,.698 linkcolorrgb.71,0.21,0.01 citecolorrgb.12,.54,.11

ansi-blackHTML3E424D ansi-black-intenseHTML282C36 ansi-redHTMLE75C58 ansi-red-intenseHTMLB22B31 ansi-greenHTML00A250 ansi-green-intenseHTML007427 ansi-yellowHTMLDDB62B ansi-yellow-intenseHTMLB27D12 ansi-blueHTML208FFB ansi-blue-intenseHTML0065CA ansi-magentaHTMLD160C4 ansi-magenta-intenseHTMLA03196 ansi-cyanHTML60C6C8 ansi-cyan-intenseHTML258F8F ansi-whiteHTMLC5C1B4 ansi-white-intenseHTMLA1A6B2

HighlightingVerbatimcommandchars=
{}

Problem 3

incolorrgb0.0, 0.0, 0.5 outcolorrgb0.545, 0.0, 0.0

breaklinks=true, colorlinks=true, urlcolor=urlcolor, linkcolor=linkcolor, citecolor=citecolor,

verbose,tmargin=1in,bmargin=1in,lmargin=1in,rmargin=1in

problem-3

4 Problem 3

[commandchars=
{}] In [1]: [rgb]0.00,0.50,0.00**from** [rgb]0.00,0.00,1.00**scipy**[rgb]0.00,0.00,1.00**.**[rgb]0.00,0.00,1.00**integrate** [rgb]0.00,0.50,0.00**import** trapz [rgb]0.00,0.50,0.00**from** [rgb]0.00,0.00,1.00**scipy**[rgb]0.00,0.00,1.00**.**[rgb]0.00,0.00,1.00**special** [rgb]0.00,0.50,0.00**import** legendre [rgb]0.00,0.50,0.00**from** [rgb]0.00,0.00,1.00**scipy**[rgb]0.00,0.00,1.00**.**[rgb]0.00,0.00,1.00**optimize**[rgb]0.00,0.00,1.00**.**[rgb]0.00,0.00,1.00**ze** [rgb]0.00,0.50,0.00**import** newton [rgb]0.00,0.50,0.00**from** [rgb]0.00,0.00,1.00**numpy**[rgb]0.00,0.00,1.00**.**[rgb]0.00,0.00,1.00**polynomial**[rgb]0.00,0.00,1.00**.**[rgb]0.00,0.00,1. [rgb]0.00,0.50,0.00**import** leggauss [rgb]0.00,0.50,0.00**import** [rgb]0.00,0.00,1.00**numpy** [rgb]0.00,0.50,0.00**as** [rgb]0.00,0.00,1.00**np** [rgb]0.00,0.50,0.00**import** [rgb]0.00,0.00,1.00**matplotlib**[rgb]0.00,0.00,1.00**.**[rgb]0.00,0.00,1.00**pyplot** [rgb]0.00,0.50,0.00**as** [rgb]0.00,0.00,1.00**plt**

Plot the 14th Legendre polynomial to eyeball the starting guesses for the zeros

[commandchars=
{}] In [2]: x [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.linspace([rgb]0.40,0.40,0.40-[rgb]0.40,0.40,0.401, [rgb]0.40,0.40,0.401, [rgb]0.40,0.40,0.4010000)

[commandchars=
{}] In [3]: l [rgb]0.40,0.40,0.40= legendre([rgb]0.40,0.40,0.4014)

[commandchars=
{}] In [4]: plt[rgb]0.40,0.40,0.40.plot(x, l(x)) plt[rgb]0.40,0.40,0.40.xlim([rgb]0.40,0.40,0.400, [rgb]0.40,0.40,0.401) plt[rgb]0.40,0.40,0.40.axhline(y[rgb]0.40,0.40,0.40=[rgb]0.40,0.40,0.400, alpha[rgb]0.40,0.40,0.40=[rgb]0.40,0.40,0.400.3, color[rgb]0.40,0.40,0.40=[rgb]0.73,0.13,0.13'[rgb]0.73,0.13,0.13bla

[commandchars=
{}] Out[4]: ¡matplotlib.lines.Line2D at 0x115eb6940¿
max size=0.90.9Problem $3_files/Problem3_{51}.png$

This has seven positive and seven negative zeros

[commandchars=
{}] In [5]: guesses [rgb]0.40,0.40,0.40= [[rgb]0.40,0.40,0.400.1, [rgb]0.40,0.40,0.400.33, [rgb]0.40,0.40,0.400.52, [rgb]0.40,0.40,0.400.7, [rgb]0.40,0.40,0.400.8, [rgb]0.40,0.40,0.400.9, [rgb]0.40,0.40,0.401.0]

newton uses the Newton-Raphson method to find zeros of a function

[commandchars=
{}] In [6]: zeros [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.array([newton(l, g) [rgb]0.00,0.50,0.00**for** g [rgb]0.67,0.13,1.00**in** guesses]) [rgb]0.00,0.50,0.00print(zeros)

[commandchars=
{}] [0.10805495 0.31911237 0.51524864 0.6872929 0.82720132 0.92843488 0.98628381]

Now I can construct the matrix of integrals for x^n

Calculate the numerical integral of x^n for even n's

[commandchars=
{}] In [7]: integrals [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.array([[rgb]0.40,0.40,0.400.5[rgb]0.40,0.40,0.40*tra x) [rgb]0.00,0.50,0.00**for** n [rgb]0.67,0.13,1.00**in** [rgb]0.00,0.50,0.00range([rgb]0.40,0.40,0.4015)[::[rgb]0.40,0.40,0.402]]) [rgb]0.00,0.50,0.00print(integrals)

[commandchars=
{}] [1. 0.33333334 0.20000001 0.14285716 0.11111114 0.09090912 0.07692312 0.06666671]

Create a matrix where each column j and row i is μ_j^{2i}

[commandchars=
{}] In [8]: functions [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.array([zeros[rgb]0.40,0.40,0.40*[rgb]0.40,0.40,0.40
[rgb]0.00,0.50,0.00**for** n [rgb]0.67,0.13,1.00**in** [rgb]0.00,0.50,0.00range([rgb]0.40,0.40,0.4015)[::[rgb]0.40,0.40,0.40

[commandchars=
{}] In [9]: np[rgb]0.40,0.40,0.40.set˙printoptions(precision[rgb]0.40,0.40,0.40=[rgb]0.40,0.40,0.401) [rgb]0.00,0.50,0.00print(functions)

[commandchars=
{}] [[1.0e+00 1.0e+00 1.0e+00 1.0e+00 1.0e+00 1.0e+00 1.0e+00] [1.2e-02 1.0e-01 2.7e-01 4.7e-01 6.8e-01 8.6e-01 9.7e-01] [1.4e-04 1.0e-02 7.0e-02 2.2e-01 4.7e-01 7.4e-01 9.5e-01] [1.6e-06 1.1e-03 1.9e-02 1.1e-01 3.2e-01 6.4e-01 9.2e-01] [1.9e-08 1.1e-04 5.0e-03 5.0e-02 2.2e-01 5.5e-01 9.0e-01] [2.2e-10 1.1e-05 1.3e-03

2.4e-02 1.5e-01 4.8e-01 8.7e-01] [2.5e-12 1.1e-06 3.5e-04 1.1e-02 1.0e-01 4.1e-01 8.5e-01] [3.0e-14 1.1e-07 9.3e-05 5.2e-03 7.0e-02 3.5e-01 8.2e-01]]

[commandchars=
{}] In [10]: np[rgb]0.40,0.40,0.40.set˙printoptions(precision[rgb]0.40,0.40,0.40=[rgb]0.40,0.40,0.408)

Get the official values to compare with

[commandchars=
{}] In [11]: mus, wts [rgb]0.40,0.40,0.40= leggauss([rgb]0.40,0.40,0.4014)

Compare the official μ values (stored in the variable `mus`) to my calculated values (stored in `zeros`)

[commandchars=
{}] In [12]: mus[[rgb]0.40,0.40,0.407:]

[commandchars=
{}] Out[12]: array([0.10805495, 0.31911237, 0.51524864, 0.6872929 , 0.82720132, 0.92843488, 0.98628381])

[commandchars=
{}] In [13]: zeros

[commandchars=
{}] Out[13]: array([0.10805495, 0.31911237, 0.51524864, 0.6872929 , 0.82720132, 0.92843488, 0.98628381])

Compare the official weights (stored in `wts`) to my calculated values (stored in `weights`)

[commandchars=
{}] In [14]: wts[[rgb]0.40,0.40,0.407:]

[commandchars=
{}] Out[14]: array([0.21526385, 0.20519846, 0.1855384 , 0.15720317, 0.12151857, 0.08015809, 0.03511946])

[commandchars=
{}] In [15]: weights [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.linalg[rgb]0.40,0.40,0.40.inv(functions[rgb]0.40,0.4
[rgb]0.40,0.40,0.40@ functions) [rgb]0.40,0.40,0.40@ functions[rgb]0.40,0.40,0.40.T [rgb]0.40,0.40,0.40@ integrals weights

[commandchars=
{}] Out[15]: array([0.2152639 , 0.20519833, 0.18553861, 0.15720292, 0.12151885, 0.08015776, 0.03511963])

Calculate the fractional error between my calculated weights and the official ones

[commandchars=
{}] In [16]: np[rgb]0.40,0.40,0.40.abs(weights [rgb]0.40,0.40,0.40- wts[[rgb]0.40,0.40,0.407:]) [rgb]0.40,0.40,0.40/ wts[[rgb]0.40,0.40,0.407:]

[commandchars=
{}] Out[16]: array([2.08311775e-07, 6.70460299e-07, 1.12400692e-06, 1.56257606e-06, 2.33478854e-06, 4.08320243e-06, 4.96419886e-06])

Pretty close! Within ˜10^{-4}%