BEPP 931 - Solution to Problem Set 5

Shasha Wang, Cung Truong Hoang, Jose Sidaoui

March 13, 2020

Question 1

Exercise 6.16: Consider $f : [1, 3]^3 \to R$ with

$$f(x_1, x_2, x_3) = (x_1 + x_2 + x_3)^{\frac{1}{4}} - (x_1 - x_2)^2 - (x_1 - 3x_3)^2 - (x_2 - 2x_3)^2 .$$

Use Matlab's lsqcurvefit to fit the neural network. Use Chebyshev interpo- lation nodes as in Algorithms 6.2 and 6.4 instead of a uniform grid to do the Chebyshev regression. Also compare tensor products and complete polynomials. How much accuracy do you lose?

Single Hidden-layer Feedforward Neural Network

First, I use a grid library to create a uniform tensor grid of 11^3 points. Then, I fit a single hidden-layer feedforward networks of the form

$$F(x; \gamma, w, b) = \sum_{j=1}^{m} \gamma_j h \left(\sum_{i=1}^{n} w_{j,i} x_i + b_j \right)$$

where $h(x) = \frac{1}{1+e^{-x}}$.

In this function, there are three variables, hence for each node we have three coefficients plus an intercept and a scalar in front of the $h(x)$ function. Altogether we have 5 coefficients for each node to solve for.

Table 1 contains the approximation errors. I use both lsqcurvefit and lsqnonlin and get exactly the same result. I also try fsolve and get approximately the same result shown in Table 2. Although I tried different initial guesses, fsolve cannot find a solution. What is worth noting is that I use "hot guesses" for other than the first fitting, so if I set the tolerance level too low, e.g. 10^{-9} (10^{-6} by default), the approximation errors will not go lower after the third iteration for fitting 3 nodes.

# nodes	# parameters	SSE	max.error
1	5	4478.049058	6.083933119
2	10	613.1201113	4.248063455
3	15	0.362644301	0.071914655
4	20	0.341402973	0.0736808
5	25	0.337219542	0.074269178
6	30	0.334773838	0.074801956
7	35	0.333759657	0.075087469
8	40	0.332817958	0.075290484
9	45	0.331976914	0.075385691
10	50	0.329159956	0.075783444

Table 1. Using lsqcurvefit/lsqnonlin - Errors of Single Hidden-layer Feedforward Neural Network

# nodes	# parameters	max.error
1	5	6.083868114
2	10	5.675767997
3	15	3.483375985
4	20	0.076443935
5	25	0.076380423
6	30	0.076380873
7	35	0.076380937
8	40	0.076381433
9	45	0.076381467
10	50	0.076381037

Table 2. Using fsolve - Errors of Single Hidden-layer Feedforward Neural Network

Chebyshev Regression

Chebyshev regression is done in two ways - tensor product and complete polynomials, for both of which I generate for every dimension 11 Chebyshev interpolation nodes using the 11 zeros of Chebyshev function of 11 orders.

For tensor product, I use grid library to generate the grid. For complete polynomials, I use grid library to generate the power matrix and use it to select polynomials accordingly.

What is worth noting is that for the method of complete polynomials, I have to do some precomputation for the values of Chebyshev function for every node, otherwise it would be too slow. For the method using the tensor grid, I use the compecon toolbox.

```
mChebyValues = zeros(nZ\_cheby\_tensor,tensorGridChebyStruct.Ndim,max(vDegreeCompleteChebyApprox)-1);

parfor chebyOrder = 1:max(vDegreeCompleteChebyApprox)

  mChebyValues(:,:,chebyOrder)=chebyshevT(chebyOrder-1,vZ\_cheby\_tensor);
```

Table 3 and Table 4 contain the approximation errors for Chebyshev regression fitting.

order	# parameters	max.error
1	8	8.5063937
2	27	0.000910483
3	64	7.41E-05
4	125	6.33E-06
5	216	5.78E-07
10	1331	5.12E-13

Table 3. Using tensor grid - Errors of Chebyshev Regression

order	# parameters	max.error
1	4	15.99250953
2	10	0.008720082
3	20	1.90E-03
4	35	4.16E-04
5	56	9.10E-05
10	286	4.60E-08

Table 4. Using complete grid - Errors of Chebyshev Regression

By comparison, we can see that Chebyshev regression can reach a much higher level of accuracy (per parameter) than neural network. Within Chebyshev regression, complete polynomial base and tensor grid seem to have the same level of accuracy per parameter, and although complete grid is supposed to be more efficient, I haven't found a way to code it so that it could beat the algorithm using the tensor grid.

Question 2

Exercise 7.5: Use midpoint rule, trapezoid rule, and Simpson's rule as well as Gauss- Laguerre quadrature.

We have the following improper integral:

$$\int_0^{\infty} e^{-\rho t} u(1 - e^{-\lambda t}) dt \qquad \textbf{Equation 1.}$$

To which we apply a change of variables to turn it into an equivalent integral over the interval $[0, 1)$, the following integral is obtained:

$$\int_0^1 e^{-\frac{x}{(1-x)}} (-e^{-a(1-e^{-\frac{\lambda x}{\rho(1-x)}})}) \frac{1}{\rho(1-x)^2} dx \qquad \textbf{Equation 2.}$$

We compute the integral using Gauss-Laguerre quadrature as well as the Midpoint, Trapezoid, and Simpson rules for the Newton-Cotes approach. The integral was computed for all possible combinations of the parameters: $a \in \{.5, 1, 2, 5\}$, $\lambda \in \{.02, .05, .1, .2\}$, and $\rho \in \{.04, .25\}$. We deal with the singularity at the node $x = 1$ by setting the $f(1) = 0$.

We display in the table below the computed integrals for all combinations of parameters using 3 nodes and one set of parameters for 5 and 10 nodes. We observe that Gauss-Laguerre converges much faster to our benchmark value for the integral (a Gauss-Laguerre approximation with 20 nodes) than the Newton-Cotes approaches, with the midpoint coming in closely after. For any particular combination of the parameters for this function, Simpson's rule seems to be the slowest in converging to the benchmark, this is particularly evident by looking at the smaller number of node formulas, in our case 3. The values Simpson gives are significantly far from the benchmark and it is only when the number of nodes is increased to 10 that this rule yields an approximation that is satisfactorily close to the rest.

# Nodes	ρ	a	λ	Gauss-Laguerre	Midpoint	Trapezoid	Simpson
3	0.04	0.5	0.02	-21.3	-20.7183	-21.3590	-24.3120
3	0.04	0.5	0.05	-19.02	-18.6065	-19.5954	-21.3331
3	0.04	0.5	0.1	-17.26	-17.0455	-18.3040	-19.6652
3	0.04	0.5	0.2	-15.86	-15.8512	-17.5099	-19.0922
3	0.04	1	0.02	-18.35	-18.0139	-18.6768	-20.7142
3	0.04	1	0.05	-14.66	-14.6280	-15.3654	-16.1823
3	0.04	1	0.1	-11.98	-12.2696	-13.4976	-13.9609
3	0.04	1	0.2	-10.07	-10.5068	-12.4425	-13.2500
3	0.04	2	0.02	-14.03	-14.0268	-14.3403	-15.3315
3	0.04	2	0.05	-8.99	-9.5304	-10.2748	-10.0535
3	0.04	2	0.1	-5.86	-6.7018	-8.2887	-8.0780
3	0.04	2	0.2	-4.07	-4.7307	-7.3542	-7.5308
3	0.04	5	0.02	-7.19	-7.8401	-8.3600	-7.5960
3	0.04	5	0.05	-2.41	-3.6107	-5.3833	-4.8589
3	0.04	5	0.1	-0.75	-1.5119	-4.5584	-4.4158
3	0.04	5	0.2	-0.27	-0.5156	-4.3506	-4.3376
3	0.25	0.5	0.02	-3.86	-3.7218	-3.9592	-4.4427
3	0.25	0.5	0.05	-3.69	-3.5692	-3.7789	-4.2507
3	0.25	0.5	0.1	-3.49	-3.3864	-3.5617	-3.9944
3	0.25	0.5	0.2	-3.23	-3.1477	-3.2994	-3.6463
3	0.25	1	0.02	-3.72	-3.5966	-3.8171	-4.3003
3	0.25	1	0.05	-3.42	-3.3201	-3.4890	-3.9401
3	0.25	1	0.1	-3.07	-3.0029	-3.1209	-3.4887
3	0.25	1	0.2	-2.64	-2.6083	-2.7063	-2.9291
3	0.25	2	0.02	-3.48	-3.3725	-3.5573	-4.0314
3	0.25	2	0.05	-2.97	-2.9101	-3.0108	-3.3974
3	0.25	2	0.1	-2.44	-2.4200	-2.4728	-2.6961
3	0.25	2	0.2	-1.83	-1.8672	-1.9369	-1.9711
3	0.25	5	0.02	-2.9	-2.8501	-2.9377	-3.3383
3	0.25	5	0.05	-2.09	-2.0985	-2.1097	-2.2520
3	0.25	5	0.1	-1.38	-1.4520	-1.5053	-1.4215
3	0.25	5	0.2	-0.71	-0.8779	-1.0467	-0.9167
5	0.04	0.5	0.02	-21.31	-21.3120	-21.4288	-21.4521
10	0.04	0.5	0.02	-21.31	-21.3146	-21.2807	-21.3346