Solution of Week 5 Lab (Prepared by Yuan Yin)

December 22, 2019

1 Error Propagation:

1.1 a.

- Prove the recursive relation (with boundary condition) and the restriction on I_n:

$\frac{x^n}{x+2} = \frac{x^{n-1}(x+2)-2x^{n-1}}{x+2} = x^{n-1} - 2\frac{x^{n-1}}{x+2}$;

$I_n = \int_0^1 \frac{x^n}{x+2}dx = \int_0^1 x^{n-1}dx - 2\int_0^1 \frac{x^{n-1}}{x+2}dx = \frac{1}{n} - 2I_{n-1}$ as required.

$I_0 = \int_0^1 \frac{1}{x+2}dx = log(x+2)\Big|_0^1 = log(\frac{3}{2}) > 0$;

Also, since $\frac{x^n}{x+2} > 0$ when $x \in [0,1] \forall n$, $I_n > 0$. Because we have $I_{n-1} > 0$ as well, $I_n = \frac{1}{n} - 2I_{n-1} < \frac{1}{n}$ as required.

- Explain the magnitude of the error you find when running BadRecurrence.m:

The output shows that $I_n = 6.0580 \times 10^{12}$ (which contradicts with the fact that $I_n < \frac{1}{n} = \frac{1}{100}$). This is because of the '-2' in the recursive relation! —— Every time when you go into the for loop, the error will be magnified by -2.

If you switch on the debug mode, you can actually see that when $n \sim 50$, I_n oscillates between negative and positive values. Why?—— Let's assume that the initial relative input error is the unit roundoff, 2^{-53}. Then, after $n = 53$ iterations, the relative error $\sim |2^{-53} \times (-2)^{53}| = 1$! Hence, when $n \sim 50$, the absolute error carried by I_n can be of the same magnitude as I_n itself! And it is now clear that we shouldn't expect this algorithm to give us any reliable answer.

- How to run the recurrence backwards?

If we try to rearrange the recursive relation, we will get $I_{n-1} = \frac{\frac{1}{n} - I_n}{2}$. Then, by guessing the initial condition to be $I_{200} = \frac{1}{200}$, we can get I_{100} through iterations.

- What do you find when running GoodRecurrence.m?

The output is $I_{100} = 0.003311185913527$. If you try to use the MATLAB command, 'integral', to check the result, you will see that GoodRecurrence.m gives a quite accurate answer.

Why? —— This is because of the '$-\frac{1}{2}$' factor in the good recursive relation! This factor means that each time when you go into the for loop, the error will be halved. Therefore, although our initial guess, $I_{200} = \frac{1}{200}$, may not be accurate enough, after 100 iterations, the absolute error will be so small that we should expect this algorithm to give us a reliable answer.

1.2 b.

- Observation:

With the decrease in the h value, |total error| firstly gets smaller until $h \sim 10^{-8}$. This optimal value of h gives min(|total error|) $\approx 10^{-8}$. After that, |total error| keeps increasing even though h gets really small.

- Explain why the error first falls as h is reduced, then rises:

In the lecture, we have proved that |total error| = |Truncation Error| + |RoundOff Error| $\leq K_1 h + K_2 \frac{u}{h} + K_3 u$. We can consider this expression as a function of h, i.e. $g(h)$. Then, working out min(| total error |) is equivalent to finding one h such that $g'(h) = 0$. However, $g'(h) = 0 \iff K_1 - K_2 \frac{u}{h^2} = 0 \iff h \approx u^{\frac{1}{2}} \approx 10^{-8}$. The truncation error dominates when $h > 10^{-8}$ while the roundoff error dominates when $h < 10^{-8}$. Also, $h \downarrow \Rightarrow R.E. \uparrow$ and T.E.$\downarrow$.

1.3 c.

1.

$I_n = \int_0^1 x^n e^{x-1} dx = [x^n e^{x-1}]\Big|_0^1 - n \int_0^1 x^{n-1} e^{x-1} dx = 1 - nI_{n-1}$ (integration by parts).

$x^n e^{x-1} > 0, \forall n$ when $x \in [0, 1] \Rightarrow I_n > 0$;

$e^{x-1} < 1$ when $x \in (0, 1) \Rightarrow I_n = \int_0^1 x^n e^{x-1} dx < \int_0^1 x^n dx = \frac{1}{n+1}$;

So, indeed $0 < I_n < \frac{1}{n+1}$.

2.

$I_{25} = 1.9279 \times 10^8$.

[1]:
```
%%file tute_5_badrecurr_PartC.m

function tute_5_badrecurr_PartC
clc

disp('illustrating numerical instability');

I_0 = 1 - exp(-1);
I = I_0;

for n = 1:25
    I = 1 - n * I;
end

disp(I);
fprintf('I must be between 0 and %6.4f!\n', 1 / (25 + 1));

end
```

```
Created file '/Users/RebeccaYinYuan/MAST30028 Tutorial Answers Yuan Yin/WEEK
5/tute_5_badrecurr_PartC.m'.
```

```
[5]: tute_5_badrecurr_PartC
```

```
illustrating numerical instability
     1.927850088325280e+08

I must be between 0 and 0.0385!
```

3.

There is something wrong with '$ - n$' in the recursive relation —— Every time when you go into the for loop, the error will be magnified by $-n$. Then using the similar arguments from the previous question, one can conclude that no matter how accurate the initial condition is, we shouldn't expect an accurate answer if we have to go inside the for loop many times.

4.

Rearranging the initial recursive relation gives: $I_{n-1} = \frac{1-I_n}{n}$.

```
[2]: %%file tute_5_goodrecurr_PartC.m

function tute_5_goodrecurr_PartC

clc

format long

I_40 = 1 / 41;
I = I_40;

for n= 40 : -1 : 26
    I = (1 - I) / n;
end

disp('using backward recurrence (stable): ');
disp(I);

disp('comparing with the true value: ');
integrand = @(x) (x.^(25)).*(exp(x - 1));
up_termin = 1;
low_termin = 0;
abs_tol = 10^(-10);
true_value = integral(integrand, low_termin, up_termin, 'AbsTol', abs_tol);
disp(true_value);

end
```



```
Created file '/Users/RebeccaYinYuan/MAST30028 Tutorial Answers Yuan Yin/WEEK
5/tute_5_goodrecurr_PartC.m'.
```

[2]:
```
tute_5_goodrecurr_PartC;
```

```
using backward recurrence (stable):
   0.037086214423739

comparing with the true value:
   0.037086214423739
```

2 Root Finding

2.1 Exercise Set 2: Fixed Point Iteration

a.

The output of 'FixedPoint.m' shows that $g_1(x)$ and $g_2(x)$ fail to find a solution while $g_3(x), g_4(x)$, and $g_5(x)$ converge to the root. Also, the convergence rate is: $g_5(x) > g_4(x) > g_3(x)$.

b.

For fixed point iteration, if it converges, the absolute error behaves like $e_{n+1} \approx ke_n$ (i.e. linear convergence), where k is different for different g (actually, $k = |g'(x_{root})|$) and $k \neq 0$. The smaller the k is, the faster the convergence. If $k = 0$, we would expect quadratic convergence.

From the outputs of 'FixedPointErrors.m', we can see that before arriving to the root, $k \approx 0.512$ for $g_3(x)$ and $k \approx 0.127$ for $g_4(x)$. For $g_5(x)$, k starts from 0.0395 and shrinks dramatically. Therefore, we can conclude that $g_5(x)$ exhibits better than linear convergence.

c.

Call 'cobweb' and try to understand the result.

d.

The last three all have the same fixed point, 1.3097995858041.

[3]:
```
%%file Ex2D.m

function Ex2D

clc
format long

N = 30;

x0_1 = 1;
x1_1 = zeros(N, 1); % preallocate memory
x1_1(1) = g1(x0_1);

for n = 2 : N
    x1_1(n)=g1(x1_1(n - 1));
```

```
end

x0_2 = 3;
x1_2 = zeros(N, 1); % preallocate memory
x1_2(1) = g1(x0_2);

for n = 2 : N
    x1_2(n)=g1(x1_2(n - 1));
end

x0_3 = 6;
x1_3 = zeros(N, 1); % preallocate memory
x1_3(1) = g1(x0_3);

for n = 2 : N
    x1_3(n) = g1(x1_3(n - 1));
end


Iterates=[x1_1 x1_2 x1_3]
disp(Iterates)

%%

clc
format long

x0 = 2;
N = 30;

x2 = zeros(N, 1); % preallocate memory
x2(1) = g2(x0);

for n = 2 : N
    x2(n) = g2(x2(n - 1));
end

x3 = zeros(N, 1); % preallocate memory
x3(1) = g3(x0);

for n = 2 : N
    x3(n) = g3(x3(n - 1));
end

x4 = zeros(N, 1); % preallocate memory
x4(1) = g4(x0);
```

```
for n = 2 : N
    x4(n) = g4(x4(n - 1));
end


Iterates=[x2 x3 x4];
disp(Iterates)
end



% ------------------------------------

% subfunctions

function y=g1(x)
y = cos(x);
end

function y=g2(x)
y = exp(exp(-x));
end

function y=g3(x)
y = x - log(x) + exp(-x);
end

function y=g4(x)
y = x + log(x) - exp(-x);
end
```



```
Created file '/Users/RebeccaYinYuan/MAST30028 Tutorial Answers Yuan Yin/WEEK
5/Ex2D.m'.
```

[7]:
```
Ex2D
```

```
   0.540302305868140  -0.989992496600445   0.960170286650366
   0.857553215846393   0.548696133603097   0.573380480369621
   0.654289790497779   0.853205311505747   0.840071952619900
   0.793480358742566   0.657571671944072   0.667409245090195
   0.701368773622757   0.791478749684416   0.785427856067595
   0.763959682900654   0.702794111808299   0.707085784986426
   0.722102425026708   0.763039187796815   0.760258236815235
   0.750417761763761   0.722738904784978   0.724658081694652
   0.731404042422510   0.749996919694713   0.748726116355687
   0.744237354900557   0.731690968525826   0.732556603421911
   0.735604740436347   0.744045681952540   0.743467047700358
   0.741425086610109   0.735734568286841   0.736126336713148
   0.737506890513243   0.741337961246103   0.741074976073316
```

0.740147335567876	0.737565726923219	0.737743288820334
0.738369204122323	0.740107770052690	0.739988350091130
0.739567202212256	0.738395886397535	0.738476414072198
0.738760319874211	0.739549242570510	0.739495036797005
0.739303892396906	0.738772423983223	0.738808955148466
0.738937756715344	0.739295741775515	0.739271141894108
0.739184399771494	0.738943248365088	0.738959822746324
0.739018262427412	0.739180701117231	0.739169538051310
0.739130176529671	0.739020754151705	0.739028274469591
0.739054790746917	0.739128498195072	0.739123432755420
0.739105571926536	0.739055921348123	0.739059333642071
0.739071365298945	0.739104810364846	0.739102511871966
0.739094407379091	0.739071878307350	0.739073426631278
0.739078885994992	0.739094061815581	0.739093018860241
0.739089341403393	0.739079118773054	0.739079821326797
0.739082298522402	0.739089184602345	0.739088711356633
0.739087042695332	0.739082404145953	0.739082722931292

1.144920592687449	1.442188102676667	2.557811897323333
1.374718782110840	1.312436529823259	3.419489986371418
1.287768285352150	1.309714605776453	4.616252276551578
1.317697367725113	1.309802422943473	6.135945666000545
1.307021814569891	1.309799491189784	7.947946196938755
1.310783209399275	1.309799588959517	10.020506365014210
1.309452110560984	1.309799585698920	12.325095516401717
1.309922438815369	1.309799585807660	14.836728547172431
1.309756162981972	1.309799585804083	17.533833952149614
1.309814935371413	1.309799585804154	20.397966310946892
1.309794160076128	1.309799585804150	23.413401514803315
1.309801503702707	1.309799585804150	26.566710087468685
1.309798907864212	1.309799585804150	29.846369019001621
1.309799825443164	1.309799585804150	33.242432210536805
1.309799501096317	1.309799585804150	36.746259349042617
1.309799615746765	1.309799585804150	40.350295783100265
1.309799575220004	1.309799585804150	44.047894508225546
1.309799589545445	1.309799585804150	47.833172061695251
1.309799584481674	1.309799585804150	51.700891436688629
1.309799586271621	1.309799585804150	55.646366460543504
1.309799585638909	1.309799585804150	59.665383243420791
1.309799585862560	1.309799585804150	63.754135250475294
1.309799585783504	1.309799585804150	67.909169299084198
1.309799585811449	1.309799585804150	72.127340365755231
1.309799585801571	1.309799585804150	76.405773538802350
1.309799585805062	1.309799585804150	80.741831801999183
1.309799585803828	1.309799585804150	85.133088604830718
1.309799585804265	1.309799585804150	89.577304385107311
1.309799585804110	1.309799585804150	94.072406373729166
1.309799585804165	1.309799585804150	98.616471140056930

3 Exercise Set 3: Bisection

Read the Tute sheet and run 'driverBisect.m'