Math 374

Midterm 1

Name: Gianluca Crescenzo

1. **Express the numbers in the forms indicated:**

   (a) 103.456_{10} in base 2, keeping 8 digits to the right of the binary point. Write this in base 16 as well;

   (b) $A0C.9_{16}$ in base 10;

   (c) Write the number -159 as a 16 bit signed integer.

Solution. I've written the following Mathematica code to help solve this problem:

```
In[1]:= ClearAll[ChangeBase];
        ChangeBase[n_, b_, fracDigits_] :=
            Module[{i, f, ibits, fbits, q, r, s, digit},
            i = IntegerPart[n];
            f = FractionalPart[n];
            ibits = "";
            fbits = "";
            While[i > 0,
              {q, r} = QuotientRemainder[i, b];
              ibits = ToString[r] <> ibits;
              i = q;];
            Do[s = b*f;
              digit = Floor[s];
              fbits = fbits <> ToString[digit];
               f = s - digit, {fracDigits}];
            If[ibits == "", ibits = "0"];
            ibits <> "." <> fbits];
```

(a) Using the `ChangeBase` module, we have:

```
In[2]:= ChangeBase[103.456, 2, 8]
        ChangeBase[103.456, 16, 8]

Out[2]= 1 1 0 0 1 1 1 . 0 1 1 1 0 1 0 0

Out[3]= 6 7 . 7 4 11 12 6 10 7 14
```

Whence $103.456_{10} = 1100111.01110100_2 = 67.74BC6A7E_{16}$

(b) We don't need to use the above code for this part. Observe that:

$$\begin{aligned} A0C.9_{16} &= 10 \cdot 16^2 + 0 \cdot 16^1 + 12 \cdot 16^0 + 9 \cdot 16^{-1} \\ &= 2572.5625. \end{aligned}$$

(c) Using the Mathematica code again gives:

```
In[4]:=  ChangeBase[159, 2, 8]

Out[4]=  1 0 0 1 1 1 1 1 . 0 0 0 0 0 0 0 0
```

So we have:

$$159 = 0000000000010011111_2.$$

Flipping each bit gives:

$$0000000000010011111_2 \rightarrow 1111111111101100000_2.$$

Finally, adding 1 gives the 16 bit signed integer expression of -159 using the 2's complement system.

$$-159 = 1111111111101100001_2.$$

2. **Complexity of Algorithms:**

   (a) An algorithm has complexity implicity expressed as $T(n) = 2T(n-1) + 4T(n-2)$. Determine $T(n)$ explicitly.

   (b) Recall the algorithm which converts a positive integer, n, from its decimal form to its binary form. Determine the complexity of this algorithm, $T(n)$.

   (c) Recall the algorithm which reduces an n by n invertible matrix to Reduced Row Echelon Form. Determine the complexity of this algorithm, $T(n)$.

Solution. (a) Rewriting the equation as $T(n) - 2T(n-1) - 4T(n-2) = 0$, and guessing that $T(n) = c_1 r^n$, we obtain:

$$\begin{aligned} & c_1 r^n - 2c_1 r^{n-1} - 4c_1 r^{n-2} = 0 \\ \implies & c_1 r^{n-2}(r^2 - 2r - 4) = 0 \\ \implies & r^2 - 2r - 4 = 0 \\ \implies & r = 1 \pm \sqrt{5}. \end{aligned}$$

Thus $T(n) = c_1(1-\sqrt{5})^n + c_2(1+\sqrt{5})^n$.

(b) The algorithm for converting a positive integer n from decimal to binary is essentially repeated divisions by 2. The complexity of this algorithm is then $T(n) = T(\frac{n}{2}) + k$, where k is the time it takes to perform a basic operation (in the case of this algorithm, it would be division or string concatenation). We solved this exact recurrence equation in Homework 2, and found that $T(n) = k\frac{\ln(n)}{\ln(2)} + c_1$.

(c) In the first row of our $n \times n$ matrix, we need to divide each of the n elements by the pivot, resulting in exactly n arithmetic operations. For each remaining $n-1$ rows, you subtract by a multiple of the pivot to zero out the first column, requiring $n^2 - n$ arithmetic operations. After handling the first column, the RREF algorithm continues on the $(n-1) \times (n-1)$ sub-matrix. Thus the complexity of the RREF algorithm is $T(n) = T(n-1) + n^2$. Observe that:

$$\begin{aligned} T(n) &= T(n-1) + n^2 \\ &= T(n-2) + (n-1)^2 + n^2 \\ &= T(n-3) + (n-2)^2 + (n-1)^2 + n^2 \\ &\vdots \\ &= T(1) + \sum_{i=2}^{n} i^2 \\ &= T(1) + \frac{n(n+1)(2n+1)}{6} - 1. \end{aligned}$$

3. **Convergence rate of iterative methods.** Recall the usual Newton's method, implemented by iterating:

$$x_{n+1} = g(x_n), \text{ where } g(x) = x - \frac{f(x)}{f'(x)}.$$

The usual convergence for this method to a nearby root of $f(x)$ was **quadratic**. Here, we present another iterative approach and ask you to evaluate it. For concreteness, we seek the value of $\sqrt{2}$. We use the following iterative method:

$$x_0 = 1.0, \quad x_{n+1} = 1 + \frac{1}{1 + x_n}.$$

   (a) Identify $g(x)$ for which $x_{n+1} = g(x_n)$. Set $a = \sqrt{2}$, and identify $g(a)$, $g'(a)$, and $g''(a)$.

   (b) Given the error of the n^{th} iteration e_n, determine the value of e_{n+1} in terms of e_n.

   (c) From the questions above, state the type of convergence achieved by this method: linear, quadratic, cubic, or other. Support your answer.

   (d) Compute the first 10 iterations, keeping 15 decimal digits. Compute also the error of each iterate, and the ratio $\frac{e_{n+1}}{e_n}$. Construct a table below.

   (e) Explain how the results of your table support your conclusions from parts (b) and (c).

Solution. (a) We have that $g(x) = 1 + \frac{1}{1+x}$. Observe that:

$$\begin{aligned} g(\sqrt{2}) &= 1 + \frac{1}{1+\sqrt{2}} \\ g'(\sqrt{2}) &= -\frac{1}{(1+\sqrt{2})^2} \\ g''(\sqrt{2}) &= \frac{2}{(1+\sqrt{2})^3}. \end{aligned}$$

(b) We should first verify that $\sqrt{2}$ is a fixed point:

$$\begin{aligned} g(\sqrt{2}) &= 1 + \frac{1}{1+\sqrt{2}} \\ &= 1 + \frac{1}{1+\sqrt{2}} \cdot \frac{1-\sqrt{2}}{1-\sqrt{2}} \\ &= 1 + \frac{1-\sqrt{2}}{-1} \\ &= \sqrt{2}. \end{aligned}$$

Now let $e_n = x_n - \sqrt{2}$. Then:

$$\begin{aligned}
e_{n+1} &= x_{n+1} - \sqrt{2} \\
&= 1 + \frac{1}{1+x_n} - \sqrt{2} \\
&= 1 + \frac{1}{1+x_n} - 1 - \frac{1}{1+\sqrt{2}} \\
&= \frac{1}{1+x_n} - \frac{1}{1+\sqrt{2}} \\
&= \frac{\sqrt{2} - x_n}{(1+x_n)(1+\sqrt{2})} \\
&= \frac{-e_n}{(1+e_n+\sqrt{2})(1+\sqrt{2})}
\end{aligned}$$

(c) Consider:

$$\begin{aligned}
\frac{e_{n+1}}{e_n} &= \frac{x_{n+1} - \sqrt{2}}{x_n - \sqrt{2}} \\
&= \frac{g(x_n) - g(\sqrt{2})}{x_n - \sqrt{2}} \\
&= \frac{\left(\sum_{k=0}^{\infty} \frac{g^{(k)}(\sqrt{2})}{k!}(x_n - \sqrt{2})^k\right) - g(\sqrt{2})}{x_n - \sqrt{2}} \\
&= \sum_{k=1}^{\infty} \frac{g^{(k)}(\sqrt{2})}{k!}(x_n - \sqrt{2})^{k-1} \\
&= \sum_{k=1}^{\infty} \frac{g^{(k)}(\sqrt{2})}{k!} e_n^{k-1}.
\end{aligned}$$

Multiplying e_n on both sides gives:

$$e_{n+1} = \sum_{k=1}^{\infty} \frac{g^{(k)}(\sqrt{2})}{k!} e_n^k$$

Thus the convergence of this iterative method is linear.

(d) The following Mathematica code computes the first 10 iterations and constructs a table:

In[5]:=
```
Clear["Global`*"]
it = 10;

vals = Table[0, {it + 1}];
err = Table[0, {it + 1}];

vals[[1]] = 1;
For[n = 1, n <= it, n++,
  vals[[n + 1]] = N[1 + 1/(1 + vals[[n]]), 1000];];
For[n = 1, n <= it + 1, n++, err[[n]] = N[vals[[n]] - Sqrt[2], 1000];];

table = Table[{
    n - 1,
    NumberForm[vals[[n]], {Infinity, 15},
     ExponentFunction -> (Null &), NumberPadding -> {"", "0"}],
    NumberForm[err[[n]], {Infinity, 15}, ExponentFunction -> (Null &),
      NumberPadding -> {"", "0"}],
    If[n == 1, "",
     NumberForm[err[[n]]/err[[n - 1]], {Infinity, 15},
      ExponentFunction -> (Null &), NumberPadding -> {"", "0"}]]
    }, {n, it + 1}];
TableForm[Prepend[table, {"n", "f[n]", "err[n]", "err[n]/err[n-1]"}]]
```

Out[5]=

n	f[n]	err[n]	err[n]/err[n-1]
0	1.000000000000000	-0.414213562373095	
1	1.500000000000000	0.085786437626905	-0.207106781186548
2	1.400000000000000	-0.014213562373095	-0.165685424949238
3	1.416666666666667	0.002453104293572	-0.172588984322123
4	1.413793103448276	-0.000420458924819	-0.171398715464729
5	1.414285714285714	0.000072151912619	-0.171602761554568
6	1.414201183431953	-0.000012378941143	-0.171567747728501
7	1.414215686274510	0.000002123901415	-0.171573755002581
8	1.414213197969543	-0.000000364403552	-0.171572724312917
9	1.414213624894870	0.000000062521774	-0.171572901151177
10	1.414213551646055	-0.000000010727040	-0.171572870810524

(e) We can see from the table that the error is increasing approximately linearly. Had it been quadratic, there would be a lot more trailing zeros. This supports our answer from part (c).

4. **Competing Species:** A famous system of equations represents two (or more) species competing for resources in a given environment. Consider, for instance, skunks and rabbits on an island competing for food, and shelter. Let $s(t)$ and $r(t)$ represent the population (in tens of thousands) of skunks and rabbits at time t. The competing species equations here are:

$$s'(t) = 6s^2 + 4rs - 20s,$$
$$r'(t) = 30r^2 + 10rs - 125r.$$

For such systems, we seek "equilibrium points". That is, values of populations of both species at which $s'(t) = r'(t) = 0$. Such points represent populations which, once achieved, remain constant. For this model, there are four such points (r, s). Use Newton's method for systems of equations to determine the one such equilibrium point for which $r \neq 0$ and $s \neq 0$.

Solution. Given the above equations, the following Mathematica code computes the first 10 iterations of Newton's method.

```
In[6]:= Clear["Global`*"]
        f1 [s_, r_] = 6 s^2 + 4 r*s - 20 s;
        f2 [s_, r_] = 30 r^2 + 10 r*s - 125 r;
        F[s_, r_] = {{f1[s, r]}, {f2[s, r]}};
        J[s_, r_] = {{D[f1[s, r], s], D[f1[s, r], r]}, {D[f2[s, r], s],
            D[f2[s, r], r]}};

        it = 6;
        sVals = Table[0, {it + 1}];
        rVals = Table[0, {it + 1}];

        sVals[[1]] = 1;
        rVals[[1]] = 4;

        For[n = 1, n <= it, n++,
          delta =
           Inverse[J[sVals[[n]], rVals[[n]]]] . F[sVals[[n]], rVals[[n]]];
          sVals[[n + 1]] = sVals[[n]] - delta[[1, 1]];
          rVals[[n + 1]] = rVals[[n]] - delta[[2, 1]];
          ];
        sol = N[Transpose[{sVals, rVals}]];

        table = Table[{
            n - 1,
            NumberForm[sol[[n]], {Infinity, 15}, ExponentFunction -> (Null &),
              NumberPadding -> {"", "0"}]}, {n, 1, Length[sol]}];

        TableForm[Prepend[table, {"n", "(s,r)"}]]
```

Out[6]=

n	(s,r)
0	{3.000000000000000, 4.000000000000000}
1	{1.824519230769231, 3.634615384615385}
2	{1.150892266024041, 3.779991223344874}
3	{0.833788952159754, 3.888824678561128}
4	{0.729275280895912, 3.923574123039542}
5	{0.714587591547749, 3.928470803795023}
6	{0.714285841760175, 3.928571386079917}
7	{0.714285714285737, 3.928571428571421}
8	{0.714285714285714, 3.928571428571429}
9	{0.714285714285714, 3.928571428571428}
10	{0.714285714285714, 3.928571428571429}

5. **Eigenvalues and Eigenvectors.**

(a) Let A have eigenvalues $\lambda_1, \lambda_2, ..., \lambda_n$, and corresponding eigenvectors $v_1, v_2, ..., v_n$. What are the eigenvalues and eigenvectors of the matrix $2A^3 - 5A + 2I$.

(b) Consider the matrix:

$$A = \begin{pmatrix} 1 & 2 & 1 \\ -2 & 9 & 3 \\ 0 & 4 & -6 \end{pmatrix}.$$

Using only the Power Method algorithm, determine all three eigenvalues and eigenvectors.

Solution. (a) Since $Av_i = \lambda_i v_i$, consider:

$$\begin{aligned}(2A^3 - 5A + 2I)v_i &= 2A^3 v_i - 5Av_i + 2Iv_i \\ &= 2\lambda_i^3 v_i - 5\lambda_i v_i + 2v_i \\ &= (2\lambda_i^3 - 5\lambda_i + 2)v_i.\end{aligned}$$

Thus for each $i = 1, ..., n$, we have that v_i is an eigenvector of $2A^3 - 5A + 2I$ corresponding to the eigenvalue $2\lambda_i^3 - 5\lambda_i + 2$.

(b) I wrote the following Mathematica code to help solve this problem:

```
In[7]:= Clear["Global`*"];
        PowerMethod[Mat_, ic_, k_, r_, it_ : 30] :=
          Module[{A, eVects, eVals, n, table, plot},
           (*Power Method*)
           A = Mat^k + r*IdentityMatrix[3];
           eVects = Table[0, {it + 1}];
           eVals = Table[0, {it + 1}];
           eVects[[1]] = ic;
           For[n = 1, n <= it, n++,
            eVects[[n + 1]] = (A.eVects[[n]])/N[Norm[A.eVects[[n]]], 1000];];
           For[n = 1, n <= it + 1, n++,
            eVals[[n]] = (eVects[[n]].(A.eVects[[n]]))/(eVects[[n]].eVects[[n]]);];
           table = Table[{
              n - 1,
              NumberForm[eVects[[n]], {Infinity, 15},
               ExponentFunction -> (Null &), NumberPadding -> {"", "0"}],
              NumberForm[eVals[[n]], {Infinity, 15},
               ExponentFunction -> (Null &), NumberPadding -> {"", "0"}],
              NumberForm[Surd[eVals[[n]] - r, k], {Infinity, 15},
               ExponentFunction -> (Null &), NumberPadding -> {"", "0"}]}, {n,
               1, it + 1}];

           (*Gershgorin Disks*)
           (*See https://resources.wolframcloud.com/FunctionRepository/
           resources/GershgorinDisks/ *)
           plot =
            Show[ResourceFunction["GershgorinDisks"][Mat],
             Graphics[{Green, PointSize[0.015],
               Point[{Surd[eVals[[it + 1]] - r, k], 0}]}], Frame -> True,
             ImageSize -> 500];
           Column[{TableForm[
              Prepend[table, {"n", "v[n]", "E[n]", "AdjustedE[n]"}]], plot},
            Spacings -> 2]
           ];
```

Since the off-diagonals of our matrix have small norm, the Gershgorin Circle Theorem says that our eigenvalues will be approximately the diagonal:

$$\lambda_1 \approx 1,$$
$$\lambda_2 \approx 9,$$
$$\lambda_3 \approx -6.$$

Since the Power Method converges to the dominant eigenvalue of A, running `PowerMethod[A,{0,1,0},1,0]` will converge to λ_2. By part (a), we saw that (non)-linear combinations of our matrix correspond to (non)-linear combinations of our eigenvalues. Considering the following shift:

$$\lambda_1 - 2 \approx -1,$$
$$\lambda_2 - 2 \approx 7,$$
$$\lambda_3 - 2 \approx -8,$$

we can see that `PowerMethod[A,{0,1,0},1,-2]` will converge to λ_3. In order to achieve convergence to λ_1, we must consider even powers of our matrix, as `PowerMethod[A,{0,1,0},1,r]` for any $r \in \mathbf{R}$ will result in convergence to either λ_2 or λ_3. By considering:

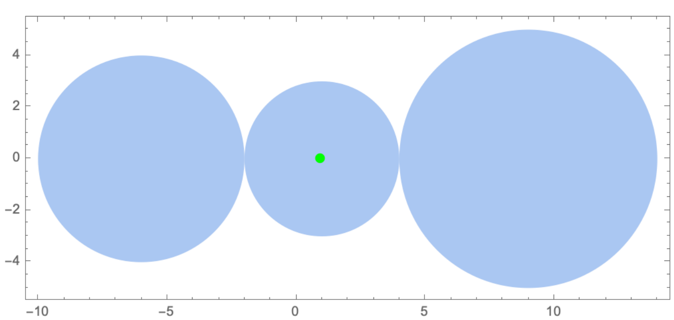
$$\begin{aligned}\lambda_1^2 - 50 &\approx -50,\\ \lambda_2^2 - 50 &\approx 31,\\ \lambda_3^2 - 50 &\approx -14,\end{aligned}$$

we have that λ_1 is now the dominant eigenvalue, whence `PowerMethod[A,{0,1,0},2,-50]` will converge to our final eigenvalue. Using such guesses, we can now obtain all of the eigenvalues and eigenvectors of A.

```
In[8]:= Mat = {{1, 2, 1}, {-2, 9, 3}, {0, 4, -6}};
        PowerMethod[Mat, {0, 1, 0}, 2, -50, 30]
        PowerMethod[Mat, {0, 1, 0}, 1, 0, 30]
        PowerMethod[Mat, {0, 1, 0}, 1, -2, 30]
```

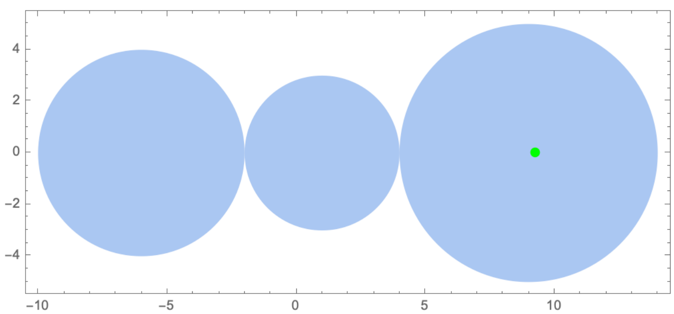
Out[8]=

n	v[n]	E[n]	AdjustedE[n]
0	{0, 1, 0}	31	9
1	{0.113914354362235, 0.882836246307320, 0.455657417448939}	31.532035685320357	9.029509160819339
2	{-0.048489646708318, 0.970658820714723, 0.235521141154687}	33.643052240993701	9.145657561979549
3	{0.186166643411090, 0.917801552016671, 0.350688311748250}	32.171910990475546	9.064872364819901
4	{-0.149218473097759, 0.946542233543928, 0.285992390465281}	31.133072508598617	9.007389883234689
5	{0.323980600484848, 0.891351744788388, 0.317062513665955}	27.557460060815695	8.806671338298920
6	{-0.339151852383970, 0.898782167907810, 0.277788832884727}	22.034572618254652	8.487318340810285
7	{0.553355926453576, 0.783281299640016, 0.283315414852338}	12.063867116105948	7.878062396053102
8	{-0.616295059126384, 0.755349797954708, 0.222771368955233}	-1.273468687339944	6.980439191960636
9	{0.805023590436778, 0.552582290044815, 0.215846777991024}	-16.227257260008820	5.811432073077270
10	{-0.849003400357572, 0.511254583127256, 0.133461520344462}	-29.346024861162986	4.544664469335114
11	{0.945733153602161, 0.294956033998363, 0.136344197477075}	-38.023502710961553	3.460707628367130
12	{-0.952288466653361, 0.299354741901435, 0.059442533426132}	-43.599592440708472	2.529902677830024
13	{0.988922365813137, 0.123944490510511, 0.081672012750008}	-46.223150932852448	1.943411708091611
14	{-0.984341079462514, 0.175426974086867, 0.017263141234959}	-47.850321699074759	1.466178127283735
15	{0.998060317309246, 0.033768553742269, 0.052299978878799}	-48.447064382170228	1.246168374590598
16	{-0.993657733812957, 0.112378769405340, -0.003914105343586}	-48.889580311421147	1.053764531847060
17	{0.999232548773546, -0.010699380951215, 0.037680720783749}	-48.999104139195087	1.000447830126545
18	{-0.996572230802665, 0.081495866512776, -0.014220145369583}	-49.126129275267079	0.934810528788011
19	{0.999011766285150, -0.032279332693873, 0.030553813258966}	-49.136082095786657	0.929471841538700
20	{-0.997600663473769, 0.066514088307211, -0.019203965563317}	-49.176773811016243	0.907318129976337
21	{0.998719659222904, -0.042716095474865, 0.027099399791679}	-49.171489648496307	0.910225439934357
22	{-0.998008331693329, 0.059264638736813, -0.021611859385393}	-49.186293157434036	0.902057006272865
23	{0.998535101837711, -0.047760528721952, 0.025427982502571}	-49.181379300808910	0.904776601814553
24	{-0.998184446207191, 0.055759280262192, -0.022775293953873}	-49.187395998755964	0.901445506530503
25	{0.998435763816338, -0.050198536398267, 0.024619757837117}	-49.184469679595838	0.903067173805006
26	{-0.998264677858654, 0.054064711201831, -0.023337522199670}	-49.187111680186256	0.901603194212256
27	{0.998385381834377, -0.051376877531138, 0.024229027932945}	-49.185568489248889	0.902458592263995
28	{-0.998302314393987, 0.053245587495216, -0.023609245813853}	-49.186783301083488	0.901785284264781
29	{0.998360476728728, -0.051946410971134, 0.024040151688464}	-49.186007321372301	0.902215428059008
30	{-0.998320238875382, 0.052849651647752, -0.023740576499156}	-49.186579968424441	0.901898016172316



Out[9]=

n	v[n]	E[n]	AdjustedE[n]
0	{0, 1, 0}	9	9
1	{0.199007438041998, 0.895533471188990, 0.398014876083996}	8.881188118811881	8.881188118811881
2	{0.258183948978711, 0.957432144129388, 0.129091974489356}	9.115264436986460	9.115264436986460
3	{0.247275274348135, 0.911683137666723, 0.328159709882572}	9.070897081737259	9.070897081737259
4	{0.261460739445668, 0.947732208250794, 0.182871384236565}	9.212482499243977	9.212482499243977
5	{0.252418998550337, 0.922950271962850, 0.290598425072376}	9.174373355153332	9.174373355153332
6	{0.259523645605768, 0.942259589826808, 0.211646740468422}	9.240175207408955	9.240175207408955
7	{0.254479280075480, 0.928629247180015, 0.269977808892033}	9.212270893116207	9.212270893116207
8	{0.258284496703984, 0.938983645725192, 0.227153762511205}	9.244056444419388	9.244056444419388
9	{0.255532040464232, 0.931536665955223, 0.258733098534404}	9.226734938682432	9.226734938682432
10	{0.257583672302553, 0.937111920420715, 0.235524734090775}	9.242785285594968	9.242785285594968
11	{0.256087755303248, 0.933059965970510, 0.252622567255688}	9.232757644603412	9.232757644603412
12	{0.257196215787952, 0.936069653298600, 0.240049392329573}	9.241105646784645	9.241105646784645
13	{0.256384565639504, 0.933869751131361, 0.249307525806369}	9.235485900933652	9.235485900933652
14	{0.256984094692745, 0.935496836881531, 0.242497099486982}	9.239905535944014	9.239905535944014
15	{0.256544090381909, 0.934303833970476, 0.247510556377275}	9.236806707694034	9.236806707694034
16	{0.256868548187901, 0.935184171209866, 0.243821891696289}	9.239170459857531	9.239170459857531
17	{0.256630126133325, 0.934537606361727, 0.246536895122187}	9.237476010462136	9.237476010462136
18	{0.256805775888522, 0.935014124069051, 0.244539120104900}	9.238747405609306	9.238747405609306
19	{0.256676615503027, 0.934663825106414, 0.246009449191200}	9.237824986414670	9.237824986414670
20	{0.256771722815388, 0.934921821256774, 0.244927480084072}	9.238510972712585	9.238510972712585
21	{0.256701762012265, 0.934732068524811, 0.245723758418121}	9.238010021968716	9.238010021968716
22	{0.256753263780350, 0.934871770973547, 0.245137784470124}	9.238380780820694	9.238380780820694
23	{0.256715371634073, 0.934768994123399, 0.245569024089579}	9.238109070583151	9.238109070583151
24	{0.256743261927753, 0.934844647012987, 0.245251673604607}	9.238309642905858	9.238309642905858
25	{0.256722739589587, 0.934788982310252, 0.245485220591750}	9.238162371949491	9.238162371949491
26	{0.256737843741756, 0.934829952093749, 0.245313351163840}	9.238270931872182	9.238270931872182
27	{0.256726729102003, 0.934799804530684, 0.245439833796355}	9.238191138626971	9.238191138626971
28	{0.256734908969926, 0.934821992156681, 0.245346753588505}	9.238249912843425	9.238249912843425
29	{0.256728889489243, 0.934805664724580, 0.245415253194369}	9.238206688593821	9.238206688593821
30	{0.256733319449663, 0.934817680806771, 0.245364843315840}	9.238238513604691	9.238238513604691



Out[10]=

n	v[n]	E[n]	AdjustedE[n]
0	{0, 1, 0}	7	9
1	{0.240771706171538, 0.842700971600384, 0.481543412343077}	6.014492753623188	8.014492753623188
2	{0.269641646334960, 0.960598365068296, -0.067410411583740}	5.878727634194831	7.878727634194831
3	{0.208909007021225, 0.788964866941862, 0.577833423675730}	4.954430729868962	6.954430729868962
4	{0.268165855112783, 0.941947872353918, -0.202042767550727}	4.426000137757763	6.426000137757763
5	{0.181446769293190, 0.699658428064928, 0.691053654900093}	3.083189300404185	5.083189300404185
6	{0.257963128785499, 0.892957869046577, -0.368891946638839}	2.025425882280533	4.025425882280533
7	{0.143420141680081, 0.572673378812293, 0.807140547959386}	0.414662886459284	2.414662886459284
8	{0.236784141893027, 0.804080062302058, -0.545333405868650}	-1.107930027478683	0.892069972521317
9	{0.098375563336622, 0.419085920823400, 0.902601373534189}	-2.561093991343339	-0.561093991343339
10	{0.206786728985371, 0.685511380739909, -0.698078359205895}	-4.145924903812512	-2.145924903812512
11	{0.053899738551656, 0.264871677635571, 0.962776096800591}	-5.090330423555879	-3.090330423555879
12	{0.174876432959983, 0.563378669313657, -0.807479230790221}	-6.350619866158883	-4.350619866158883
13	{0.016422475713700, 0.133227509307199, 0.990949409937376}	-6.791442682731066	-4.791442682731066
14	{0.147051461584500, 0.458887317108476, -0.876241004429060}	-7.633491232517589	-5.633491232517589
15	{-0.011921788970571, 0.032695803353685, 0.999394244225370}	-7.766152446784125	-5.766152446784125
16	{0.125524885459548, 0.378996535521444, -0.916845204597325}	-8.282583895207775	-6.282583895207775
17	{-0.032088853814853, -0.039336806078057, 0.998710629335858}	-8.276631457407656	-6.276631457407656
18	{0.109915183408621, 0.321500039053050, -0.940508573775453}	-8.584987415300720	-6.584987415300720
19	{-0.046011244281938, -0.089313071266811, 0.994940269915997}	-8.533335025748374	-6.533335025748374
20	{0.098965690526613, 0.281366432248799, -0.954483484876597}	-8.718312913381825	-6.718312913381825
21	{-0.055493771604990, -0.123470590520135, 0.990795364638765}	-8.661089881308980	-6.661089881308980
22	{0.091412992297620, 0.253772710919455, -0.962934616696163}	-8.774087253282586	-6.774087253282586
23	{-0.061917119573245, -0.146664272085582, 0.987246606272900}	-8.725191861845075	-6.725191861845075
24	{0.086247832096551, 0.234942417321101, -0.968175279586275}	-8.795721191349795	-6.795721191349795
25	{-0.066259985463891, -0.162371628496872, 0.984502447221742}	-8.758019091317310	-6.758019091317310
26	{0.082731321910357, 0.222140940843063, -0.971498291700157}	-8.802929884371053	-6.802929884371053
27	{-0.069194976190985, -0.172998966762820, 0.982488377930714}	-8.775324328962737	-6.775324328962737
28	{0.080343066694655, 0.213455151444354, -0.973643615475378}	-8.804398647960546	-6.804398647960546
29	{-0.071178720009127, -0.180187461529019, 0.981053550284382}	-8.784767287259034	-6.784767287259034
30	{0.078723296203653, 0.207568075760550, -0.975047761168598}	-8.803831393957733	-6.803831393957733