Appendix III: Computer Algebra System Codes

The lines of *Mathematica* code in this appendix implement various algorithms as presented in the text. If you use a different computer algebra system, many of these code modules can be adapted to your system.

Code 0: Mathematica syntax. In this section we alert the user to frequently-used Mathematica syntax that may be different in other CASs.

Comments. Surround comments with `(* ... *)`.

Lists. Use `L[[i]]` to refer to element `i` in list `L`. If a list has an element that itself is a list, two indices are needed. For example, if `L1` is `{{2,3},{5,3}}`, then `L1[[2]]` refers to the list `{5,3}`, whereas `L1[[2,1]]` refers to the element 5.

Matrices. The 2×2 matrix $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ is stored as a list of lists: `mat = {{a,b},{c,d}}`. To multiply matrices, use the operation Dot (.), as in `mat.mat`. If `V` is a vector, `V.V` gives the dot product. If `V` has length 2, Mathematica automatically uses the column vector for V in the calculation of the matrix product `mat.V`.

Logical operators. Use `&&` for "and" and `||` for "or".

Conditional statements. The phrase `If[ expr, stmt ]` executes statement `stmt` if the expression `expr` is true. Furthermore, `If[ expr, stmt1, stmt2 ]` is interpreted as, If `expr` then `stmt1` else `stmt2`. In statement construction, any `stmt` may be a single statement or a sequence of statements separated with semicolons.

Do loop. The expression `Do[ stmt, {i, imin, imax} ]` executes `stmt` for integer `i` values `imin` to `imax`.

Block. A helpful trick in function construction is to use local variables so that calls to the function avoid unintended consequences. One way to implement this trick is to place the code for a function within a `Block`, whose syntax is `Block[ { varlist }, stmt ]`. The variables in `varlist` are local to the block and are separated with commas. These variables may be initialized within the list, such as, for example, `Block[ { a, b, c = 5 }, ... ]`.

Functions. Define functions using `funcname[ inputvariablelist ] := body;`. By custom, user-defined function names usually start with a lower-case letter so as to avoid conflicts with standard Mathematica commands and functions. The input variable list determines the variables to be given in a call to the function. Each variable name must end in an underscore. For example, `pattern[ L_,R_ ] := Block[ body ];`. In the body of the function, variables `L` and `R` are used without the underscore. ♢

Code 1: Built-in functions. Many of the Mathematica commands we use in the codes of this appendix may be standard functions or routines that have similar counterparts in the CAS of your choice. Here are a few examples.

- To find the decomposition of any given integer n as a product of powers of its prime divisors, use `FactorInteger`.

  ```
  L1 = FactorInteger[n] (*L1 is the output of this list*)
  ```

  When `n = 1000`, `FactorInteger` returns `L1` as {{2, 3}, {5, 3}}, which is interpreted as $2^3 * 5^3$.
- To find $\phi(n)$, the Euler phi function or the totient of Chapter I, use `EulerPhi[n]`. For example, `EulerPhi[10]` returns the integer 4.
- `Prime[n]` returns the n^{th} prime. For example, `Prime[4]` returns 7.
- `PrimeQ[n]` tests whether n is prime, returning `True` if it is prime, and `False` otherwise.
- `BaseForm[m,n]` writes the integer m in base n. Thus `BaseForm[45,2]` returns $(101101)_2$.
- `FromDigits[string,m]` accepts a string of alpha-numeric symbols in base `m` and returns the value as a decimal integer. For example, `FromDigits["1AB",16]` returns the decimal number 427. ♢

Code 2: Nim addition. The following functions perform nim addition. The function `nimAdd` outputs the nim sum of two nonnegative integers.

```
nimAdd[m_, n_]:= Block[{k = 1,y = Min[{m,n}], z = Max[{m,n}], sum=0},
   While[k < z, k = 2*k];            (*obtain power of 2 at least z*)
   While[k >= 1,                                      (*loop       *)
      If[(y<k && k<=z)||(z<k && k<=y), sum = sum + k];(*update sum *)
      If[k <= z, z = z - k]; If[k <= y, y = y - k];   (*update z, y*)
      k = k/2];                                       (*update k   *)
   sum]                                               (*return sum *)
```

For example, `nimAdd[2, 3]` returns 1.

`NimList[X]` outputs the nim sum for a list `X` of at least two nonnegative integers.

```
NimList[X_]:= Block[{n = Length[X], sum= X[[1]]},(*sum: term 1 of X*)
   Do[sum = nimAdd[sum, X[[i]]], {i, 2, n}];   (*update sum in loop*)
   sum]                                        (*return sum        *)
```

For example, `NimList[{5,7,12}]` gives the nim sum 14. ♢

Code 3: Mancala nim. The following functions determine the nim value of a mancala-nim board. `Mex[X]` returns the minimal excluded value for a list `X` of nonnegative integers.

```
Mex[X_]:= Block[{mex = 0, i, flag = False},
  While[ Not[flag],                              (*loop            *)
     If[ Not[ MemberQ[X, mex] ],                 (*If true         *)
         flag = True,                            (*the mex is found*)
         mex = mex + 1]];                        (*else update mex *)
 mex];                                           (*return mex      *)
```

For example, `Mex[{0,1,2,4,5}]` returns the value 3.

`NMove[X]` generates the next possible configurations from a list `X` corresponding to the number of stones in the basins of a mancala board.

```
NMove[X_]:= Block[{move = {}, i, j, Y, m = Length[X]},
 If[m>0,
      Do[ If[ X[[i]] == i,                      (*if pit i is hot   *)
         Y = X; Y[[i]] = 0;                     (*then make the move*)
```

```
         Do[ Y[[j]] = Y[[j]] + 1,{j, 1, i-1 }];(*update Y          *)
         move = Append[move, Y]], {i, 1, m}]]; (*update move       *)
 move];                                        (*return move       *)
```

For example, executing `NMove[{1,2,3,5,0,0,7}]` generates the configurations $\{\{0,2,3,5,0,0,7\},\{2,0,3,5,0,0,7\},\{2,3,0,5,0,0,7\},\{2,3,4,6,1,1,0\}\}$.
`nimValue[X]` computes the nim value of a mancala board X.

```
nimValue[X_]:= Block[{moves = NMove[X], values, m, i},
   m = Length[moves];
   If[moves == { },                           (*If there are no moves  *)
       0,                                     (*return 0               *)
       values= {};                            (*else continue          *)
          Do[values = Append[values, nimValue[ moves[[i]] ]],(*loop*)
             {i, 1, m}];                      (*end loop               *)
       Mex[values]]                           (*return mex value       *)
 ];
```

For example, `nimValue[{1,2,3,4,5,6}]` returns the nim value 3. To evaluate the nim sum value for a collection of mancala nim boards, evaluate each board with the algorithms of this section, and then add the nim values of the boards using the appropriate command from Code 2. ♢

Code 4: The greatest common divisor. To find the greatest common divisor d of two positive integers a and b and to write d as a linear combination of a and b, use `ExtendedGCD[a,b]`. For example, `ExtendedGCD[5,17]` gives $\{1, \{7, -2\}\}$. So we can write $1 = 7 \cdot 5 - 2 \cdot 17$.

To solve the equation $ax \equiv b \bmod c$, given that $\gcd(a,\ c) = 1$, write 1 as a linear combination of a and c as in the paragraph above, which gives the multiplicative inverse of a. For example, the solution of $5x \equiv 3 \bmod 17$ is obtained by multiplying through by 7, the multiplicative inverse of 5 modulo 17. This multiplication results in $35x \equiv 21 \bmod 17$, which simplifies to $x \equiv 4 \bmod 17$. That is, the primitive solution to the equation is $x = 4$. ♢

Code 5: The Mandelbrot set. The command `MandelbrotSetPlot[{−1.5 −I, 0.6 + I}]` produces the Mandelbrot set of Figure IV.5a. The arguments for this command are the complex coordinates of the lower left-hand and upper right-hand corners of a window onto the Mandelbrot set. This particular Mathematica command may have some counterpart in the CAS of your choice. ♢

Code 6: From tree addresses to fractions via matrices. The following *Mathematica* code for Proposition IV.45 generates the fraction $\frac{p}{q}$ residing at address w in the Stern-Brocot tree of Chapter IV.

```
f[{{a_, c_}, {b_, d_}}] := (a + c)/(b + d);    (*array to fraction *)

  (*     Define the four 2x2 matrices from Proposition IV.45.      *)
rl = {{0, 1}, {1, 2}}; lr = {{1, 1}, {2, 1}};  (*two arrays        *)
l = {{1, 1}, {0, 1}}; r = {{1, 0}, {1, 1}};    (*two more arrays   *)
```

With address `llrrlrl`, `f[rl.l.r.r.l.r.l]` returns $\frac{13}{44}$. Remember that the argument to the function `f` is a product of matrices. ♢

Code 7: The simplest number in an interval, from Exercise IV.5. The following code is a way to find the simplest number in `[L,R]` where `0 < L < R`

< 1 with R and L real numbers. The function call `simple[L,R]` returns a 4-tuple: {m, n, m/n, S} where m/n is a fraction in the interval; S is True if L < m/n < R and False otherwise. With integers p and q, 0 < p < q, `left[p,q]` returns {a, b, c, d, a/b, c/d} where a/b and c/d are the left-parent and right-parent, respectively, of fraction p/q. With p/q in [L,R], `mostL[p,q,L,R]` and `mostR[p,q,L,R]` return 3-tuples: {a, b, a/b} where a/b is, respectively, the simplest fraction in interval [L, R] left or right of p/q.

```
simple[L_, R_] := Block[{n = Ceiling[1/(R-L)], m, d},
   m=Ceiling[n L];                      (*m/n is in [L, R]       *)
   d=GCD[m, n]; m = m/d; n = n/d;       (*reduce m/n             *)
   {m, n, m/n, L <= m/n <= R}           (*return this list       *)
   ];
left[p_, q_] := Block[{a, b, c, d, ee, LR}, ee = ExtendedGCD[p, q];
   a = Abs[ee[[2, 2]]]; b = Abs[ee[[2, 1]]]; c = p - a; d = q - b;
   If[a/b < p/q,
      LR = {a, b, c, d, a/b, c/d},      (*If true, return list,  *)
      LR = {c, d, a, b, c/d, a/b}]      (*Else, return this list.*)
   ];
mostL[p_, q_, L_, R_] := Block[{n, m, pp = p, qq = q},
   {n, m} = {left[p, q][[1]], left[p, q][[2]]};
   If[L <= n/m <= R,
      mostL[n, m, L, R],                (*If true, recur.        *)
      {pp, qq, pp/qq}]                  (*Else, return this list.*)
   ];
mostR[p_, q_, L_, R_] := Block[{n, m, pp = p, qq = q},
   {n, m} = {left[p, q][[3]], left[p, q][[4]]};
   If[L <= n/m <= R,
      mostR[n, m, L, R],                (*If true, recur.        *)
      {pp, qq, pp/qq}]                  (*Else, return this list.*)
   ];
```

The input simple[Sqrt[10]/5, Sqrt[2]/2] returns $\{9, 14, \frac{9}{14}, \text{True}\}$.
The input mostL[9, 14, Sqrt[10]/5, Sqrt[2]/2] outputs $\{7, 11, \frac{7}{11}\}$.
The input mostR[9, 14, Sqrt[10]/5, Sqrt[2]/2] outputs $\{2, 3, \frac{2}{3}\}$.
Thus the simplest number in this interval is $\frac{2}{3}$. ◊

Code 8: Generating regular Babylonian numbers between 1 and 3. The following code generates a list of the fractions $\frac{p}{q}$ where $2 + \frac{p}{q}$ is no more than a third order fraction and its reciprocal is no more than a fourth order fraction. A slight modification of the function will do the same for fractions of the form $2 - \frac{p}{q}$; in particular, in the third line of the code, change both plus signs to negative signs, and in the fourth line, change the first plus sign to a negative sign.

```
regular = Block[{ fractions = { }, n},
   Do[                                 (*begin loop              *)
       If[1/(2 + n/60^3)*60^4 == Floor[ 1/(2 + n/60^3)*60^4 ] &&
          2 + n/60^3 < 1 + Sqrt[2],
          fractions = Append[ fractions, n ]],
             {n, 0, 60^3-1} ];         (*end loop                *)
   fractions/60^3]                     (*return list of fractions*)
```

The output of function `regular` returns the fourteen terms of $\widehat{A}$ as given on p. 143 along with 0 (except in reverse order): $\{0, \frac{1}{40}, \frac{6}{125}, \frac{1}{12}, \ldots, \frac{2}{5}\}$. ◊

Code 9: Generating patterns related to Puzzle VII.3. The following code calls several functions from Code 7. Given real numbers L and R with $0 < L < R < 1$, the output from the function call pattern[L, R] is a list of two integers $\{N, M\}$ which means that $\frac{N}{M}$ is the simplest fraction in $[L, R]$.

```
pattern[L_, R_] :=
  Block[{s = simple[L, R], ll, rr},
   ll = mostL[s[[1]], s[[2]], L, R];     (*ll is simplest on left  *)
   rr = mostR[s[[1]], s[[2]], L, R];     (*rr is simplest on right *)
   If[ll[[2]] <  rr[[2]],                (*If ll simpler than rr   *)
     {ll[[1]], ll[[2]]},                 (*Then return fraction ll *)
     {rr[[1]], rr[[2]]}]                 (*Else return fraction rr *)
   ];
seePattern[p_] :=
   ListPlot[Table[{j, pattern[(j - 1)/p, (j + 1)/p][[2]]/p},
      {j, 2, p - 2}]]
```

Upon executing `seePattern[71]`, the data generated in the Table appears as Figure VII.19a, each point of which is an ordered pair whose second component is the ratio of the denominator (of the simplest fraction over a specified interval of length $\frac{2}{71}$) and 71. ◊

Code 10: Generating good approximations. This code generates a list of the fractions $\frac{p}{q}$ up to `q = m` that are good, reduced approximations to `omega`.

```
good[m_, omega_] := Block[{approx = {}, j, p},
  Do[ p =  Round[n * omega];
     If[GCD[p, j] == 1 && Abs[omega - p/j] < 1/j^2,
        approx = Append[approx, Round[j * omega]/j]], {j, 1, m}];
  approx]
```

The output for `good[1000, Pi]` is $\{3, \frac{19}{6}, \frac{22}{7}, \frac{333}{106}, \frac{355}{113}\}$. ◊

Code 11: Generating signatures of irrational numbers. The following code generates a graph of points of the form `{n, Sin[2*Pi*omega*n]}` as `n` ranges from 0 to `m` where `omega` is a given irrational number.

```
sig[ omega_, m_ ] := Block[ {n},
  ListPlot[Table[ {n, Sin[2*Pi*omega*n]},    (*generate a table     *)
    {n, 0, m}]]]                             (*and plot the points  *)
```

The output for `sig[Pi, 800]` is Figure IX.14b (without strand 0 highlighted). ◊

Code 12: The harmonic algorithm, H. The following code generates the harmonic-continued-fraction-like convergents from Chapter VII for the positive real number x, with m being the initial number of strands (which we often take to be 1), and k is the desired number of convergents.

```
hcf[x_, m_, k_] := Block[{ q = m, p, rr, n, strands = {m}, e = 1,
      convergents = {}, d, j = 1, flag = True },
   While[ flag, p = Round[ x*q ];         (*begin loop              *)
      d = GCD[p, q];
      p = p/d; q = q/d;
      If[ q*x - p > 0, e = 1, e = -1 ];
      convergents = Append[ convergents, p/q ];
      If[ q*x == p || j == k, flag = False ];
      j = j + 1;                            (*increment counter      *)
      If[ q > 1, rr = Mod[ -e*ExtendedGCD[p, q][[2, 1]], q], rr = 0];
      If[Abs[ q*x - p] > 0,
         n = Round[ (rr*(x*q - p) - e)/(q*(p - x*q))];
      strands = Append[ strands, n ]; q = Abs[ q*n + rr] ]
   ];                                       (*end loop               *)
   { strands, convergents }];               (*return lists           *)
```

The input `hcf[Pi,1,5]` produces the output

{{1, 7, 16, 293, 2, 3}, {3, 22/7, 355/113, 104348/33215, 312689/99532}},

where the first list consists of successive integers `n` where the next value is `q*n+rr` from Proposition VII.21 and the second list consists of the convergents of π. ♢

Code 13: Chapter VIII, Proposition 13: Newton's ratios. The following code can be used to check a variety of computations from Chapter VIII. To condense the syntax, we use `p` and `w` in place of ρ and θ, respectively. The constant `k1` is a conversion from miles to meters; `T1` is Earth's period in seconds. Function `A1` computes $A(\rho, R)$ of Equation (VIII.10), and `a1` is A's helper function $\alpha(\rho, R)$ defined prior to Proposition VIII.16. Functions `B1, C1, and h1` respectively compute B of Equation (VIII.13), C of Equation (VIII.15), and h of Puzzle VIII.25.

```
G1 = 6.67*10^-11; M1 = 5.98*10^24; (*Gravity constant & Earth mass *)
a1[p_, R_]:= G1*M1/(4/3* Pi*p^2*R); T1 = 86164;  k1 = 1609;
gp = 9.828; ge = 9.785;             (*gravity at pole and equator *)
A1[p_, R_]:= If[p > R, 2*Pi*R*a1[p, R]*(2*p^2/(p^2 - R^2)
   - p^2*R/(p^2-R^2)^(3/2)(Pi/2 - ArcSin[(2*R^2 - p^2)/p^2])),0];
Q1[p_, R_, w_]:= p^2/R^2 *Cos[w]^2 + Sin[w]^2;
B1[p_, R_]:= If[p > R, 8*p*a1[p,R]*NIntegrate[1/(Q1[p, R, w]-1)
   *(-1 + Sqrt[Q1[p, R, w]/(Q1[p, R, w] - 1)]*Log[Sqrt[Q1[p, R, w]]
   + Sqrt[ Q1[p, R, w] - 1 ] ]), {w, 0, Pi/2}], 0];
C1[p_, R_]:= If[p > R, B1[p, R] - p*(2*Pi/T1)^2, 0];
h1[p_, R_]:=Sqrt[(A1[k1*p, k1*R] - gp)^2 + (c[k1*p, k1*R] - ge)^2];
```

To check Proposition VIII.19, enter `A1[101,100]/B1[101,100]`. To generate Figure VIII.22, enter

```
Plot[{A1[3971*k1,R*k1]/C1[3971*k1,R*k1],230/229.},{R,3950,3975}].
```

```
ContourPlot[h1[p, R], {p, 3960, 3985 }, {R, 3918, 3960},
   Contours-> {.01, .005 }, PlotPoints -> 20]
```

To generate Figure VIII.23, the `ContourPlot` command above may take about a minute to execute. To shorten the time, use a smaller value for `PlotPoints`, such as `PlotPoints -> 3`. ♢

Code 14: Chapter VIII, Proposition 30: Vindicating Newton. The function `phi` is the parameter function ϕ, defined following Definition VIII.26. The input variable `theta` for `phi` is in degrees. The functions `phi1, phi2, phi3` are the respective functions ϕ_1, ϕ_2, ϕ_3. The function `Qe` and `Qe` are respectively Equations (VIII.17) and (VIII.18). The execution time for this code may be several minutes. To shorten the execution time, use a smaller value for `PlotPoints`, such as PlotPoints->3. For simplicity, we use p in place of ρ. To adapt this code to solve the same problem using latitudes other than 66, 67, and ± 0.5, change those numbers in the code. You may need to define a fourth parameter function, q4[p,R], and modify Qe appropriately.

```
phi[p_, R_, theta_] := ArcTan[R*Tan[Pi*theta/180]/p];(*radian mode*)
phi1[p_, R_]:= phi[p, R, 66.]; phi2[p_, R_] := phi[p, R, 67.];
   phi3[p_, R_] := phi[p, R, 0.5];
Qa[p_, R_] := NIntegrate[Sqrt[p^2*Sin[u]^2 + R^2*Cos[u]^2],
   {u, phi1[p, R], phi2[p ,R]}];          (*u is a dummy variable *)
Qe[p_, R_] := 2*NIntegrate[Sqrt[p^2*Sin[u]^2 + R^2*Cos[u]^2],
   {u, 0, phi3[p, R]}];
M[p_, R_] := Sqrt[ (Qa[p, R] - 69.52)^2 + (Qe[p, R] - 68.76)^2 ];

ContourPlot[ M[p, R], {p, 3973.5, 3975}, {R, 3956, 3957.5},
   Contours->{.01, 0.005}, PlotPoints -> 20 ]
```

Code 14 generates Figure VIII.28. ◊

Code 15: Chapter VIII, Puzzle 32: Mercury and Earth. In the following code, we use Q instead of θ. The function time[Q,e,T] is $\tau_{e,T}(Q)$, Equation (VIII.25). The function Qm[t] uses Newton's method to solve the equation time[Q,0.21,0.241] == t (where we use a double equal sign for equation equality) for Q given the initial guess Q =2*Pi*t/0.241 since the solution Q should be near the Q solution of the equation t == 0.241*Q/(2*Pi). The value of Qmercury is a list of $(13 \cdot 365 + 1)$ numbers. The semi-colon at the end of the Table command suppresses output display on the computer screen. The function rm[Q] is Equation (VIII.23) with respect to Mercury.

```
time[Q_, e_, T_] :=
   T/(2*Pi)*(Q - 2*ArcTan[e*Sin[Q]/(1 + Sqrt[1 - e^2] + e*Cos[Q])]
      - e*Sqrt[1 - e^2]*Sin[Q]/(1 + e*Cos[Q]));
Qm[t_] :=  FindRoot[time[Q, 0.21, 0.241] == t,
   {Q, 2*Pi*t/0.241 }][[1, 2]];
Qe[t_] := FindRoot[time[Q, 0.017, 1] == t,
   {Q, 2*Pi*t}][[1, 2]];
Qmercury =  Table[ Qm[j/365], {j, 0, 365*13}];
Qearth   =  Table[ Qe[j/365], {j, 0, 365*13}];
rm[Q_] := (1 - 0.21^2)*0.387/(1 + 0.21*Cos[Q]);
re[Q_] := (1 - 0.017^2)/(1 + 0.017*Cos[Q]);
Z[Q_, omega_] :=  re[Q]*{Cos[Q], Sin[Q]}
   - rm[omega]*{Cos[omega], Sin[omega]};
d1[Q_, omega_] := Sqrt[Z[Q, omega].Z[Q, omega]];

1/(13* 365) Sum[d1[ Qearth[[j]],  Qmercury[[j]] ], {j, 1, 13*365}]
```

The result in executing this last command is 1.03903 AU, the average distance between Earth and Mercury over a thirteen year span (sampling the distance between them once a day). ◊

Code 16: Fractal trees. The following code will generate trees. In the function call tree[n,w,s], n is the number of levels, w is the angle at which the branches

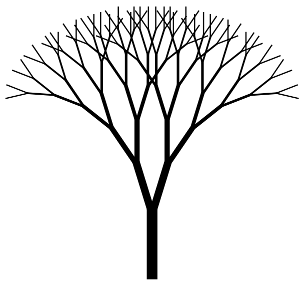

FIGURE 3. A tree with seven branch levels.

bend, and s is a scaling factor between the branch levels. In many CAS systems, a 2×2 array is a list of the rows of the matrix. Thus the matrix $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ is encoded as A={{a,b},{c,d}}. For ease of reading, in this code we use the array format rather than the list format for matrices.

$$\texttt{bL[X\_, w\_, s\_]:=\{X[[2]], X[[2]] + s} \begin{bmatrix} \texttt{Cos[w]} & -\texttt{Sin[w]} \\ \texttt{Sin[w]} & \texttt{Cos[w]} \end{bmatrix} \texttt{.(X[[2]] - X[[1]])\};}$$

(*branch left *)

$$\texttt{bR[X\_, w\_, s\_]:=\{X[[2]], X[[2]] + s} \begin{bmatrix} \texttt{Cos[w]} & \texttt{Sin[w]} \\ -\texttt{Sin[w]} & \texttt{Cos[w]} \end{bmatrix} \texttt{.(X[[2]] - X[[1]])\};}$$

(*branch right *)

```
tree[n_, w_, s_]:= Block[{leaves, twig, i, j, buds, m = 1},
   leaves = { {{0, 0}, {0, 1}} };      (*initial endpoints of twigs*)
   twig = {Thickness[.05], Line[ leaves[[1]] ]}; (*trunk of tree   *)
   Do[ buds ={} ;                            (*loop              *)
      twig = Append[twig, Thickness[.05*0.7^j]]; (*twig thickness  *)
      Do[buds=Append[buds, bL[leaves[[i]], w, s]]; (*grow left bud *)
         buds=Append[buds,bR[leaves[[i]],w,s]],{i,1,m}]; (*rt. bud *)
      leaves = buds; m = Length[ leaves ];
      Do[twig=Append[twig,Line[leaves[[i]]]], {i,1,m}]; (*grow twig*)
       {j, 1, n}];                  (*update leaves    *)
   Graphics[twig]];                            (*graph the tree  *)
```

Figure 3 was generated by tree[7,0.3,0.85]. ◊

Code 17: The Regular continued fraction. This function accepts a real number omega and a positive integer max and uses the Regular continued fraction algorithm (R) to produce a list of partial denominators nk and convergents Ck for 0 <= k <= max.

```
rcf[omega_, max_] := Block[{a = 1, b = 0, c = Floor[omega],
      d = 1, denominators, convergents,
      k=0, pk, qk, nk, sk, flag = True},
   denominators = {c}, convergents = {c},
   While[ flag,
      k = k+1;
      sk = (a - omega*b)/(omega*d - c);
      nk = Floor[sk];
      pk = a + nk*c;  a = c;  c = pk;             (* advance p, q  *)
      qk = b + nk*d;  b = d;  d = qk;
      denominators = Append[denominators, nk];    (* update lists  *)
      convergents  = Append[convergents, pk / qk];
      If[ nk == sk || k == max, flag = False ]    (* check if done *)
      ];                                          (* end of loop   *)
   {denominators, convergents,                    (* print output  *)
       1.0*convergents} ]                         (* decimal approx*)
```

The input `rcf[E,3]` produces
$\{\{2, 1, 2, 1\}, \{2, 3, 8/3, 11/4\}, \{2., 3., 2.66667, 2.75\}\}$. ♢

Code 18: The nearest integer continued fraction, Z**.** This function accepts a real number `omega` and a positive integer `max` and uses the Nearest Integer continued fraction algorithm (Z) to produce a list of partial denominators `nk` and convergents `Ck` for `0 <= k <= max`.

```
zcf[omega_, max_] := Block[{a = 1, b = 0, c = Round[omega],
      d = 1, denominators, convergents,
      k=0, pk, qk, nk, sk, ek, flag = True},
   denominators = {c}, convergents = {c},
   While[ flag,
      k = k+1;
      sk = (a - omega*b)/(omega*d - c);
      ek = Sign[sk];
      nk = Round[ Abs[sk] ];
      pk = ek*a + nk*c;  a = c;  c = pk;         (* advance p, q   *)
      qk = ek*b + nk*d;  b = d;  d = qk;
      denominators = Append[denominators, ek*nk];  (* update lists *)
      convergents  = Append[convergents, pk / qk];
      If[ sk == Round[sk] || k == max, flag = False ]  (*Done?     *)
      ];
   {denominators, convergents,                   (* print output   *)
    1.0*convergents} ]                           (* decimal approx *)
```

The input `zcf[Pi, 3]` produces
$\{\{3, 7, 16, -294\}, \{3, 22/7, 355/113, 104348/33215\}, \{3., 3.14286, 3.14159\}\}$. ♢

Code 19: The Greedy continued fraction, G**.** This function accepts a real number `omega` and a positive integer `max` and uses the Greedy continued fraction algorithm (G) to produce a list of partial denominators `nk` and convergents `Ck` for `0 <= k <= max`. This code implements Proposition IX.26.

```
gcf[omega_, max_] := Block[{a = 1, b = 0, c = Round[omega],
      d = 1, denominators, convergents,
      k=0, pk, qk, nk, sk, ek, flag = True},
   denominators = {c}, convergents = {c},
   While[ flag,
      k = k+1;
      sk = (a - omega*b)/(omega*d - c);
      ek = Sign[sk];

      tsk = Floor[Abs[sk]];
      nk = Ceiling[ Abs[sk] - (b + ek*d*tsk)/(2*b+ek*d*(2*tsk+1)) ];
      pk = ek*a + nk*c;  a = c;  c = pk;           (* advance p, q *)
      qk = ek*b + nk*d;  b = d;  d = qk;
      denominators = Append[denominators, ek*nk];  (* update lists *)
      convergents  = Append[convergents, pk / qk];
      If[ sk == Round[sk] || k == max, flag = False ]  (*Done?     *)
      ];
   {denominators, convergents,                     (* print output *)
    1.0*convergents} ]                             (*decimal approx*)
```

The input `gcf[EulerGamma, 3]` produces
$\{\{1, -2, 3, 4\}, \{1, 1/2, 4/7, 15/26\}, \{1., 0.5, 0.571429, 0.576923\}\}$. ♢

Code 20: The mean-value rule of Algorithm G. This code is useful with respect to the proof of Proposition IX.26. The pertinent values for variable `e` in this code are ± 1.

```
meanG[a_, b_, c_, d_, e_] := Block[{m, delta, f},    (*e is 1 or -1*)
    f[j_] := (a * j + c)/(b*j + d);
    Solve[2*f[m + e*delta] == f[m] + f[m + e], delta]];

meanG[a, b, c, d, 1]
```

Executing this code generates the solution, {{`delta->(d+bm)/(b+2d+2bm)`}}. Furthermore, executing the command

```
meanG[a, b, c, d, -1]
```

generates the solution {{`delta->(d+bm)/(-b+2d+2bm)`}}. ♢

Code 21: Factoring integers via continued fractions. For Proposition IX.35 we adapt the function `rcf` from Code 17, renaming it `Rcf`, so as to output two additional lists: the numerators, and the denominators of the convergents of x.

```
Rcf[omega_, max_] := Block[{a = 1, b = 0, c = Floor[omega],
      d = 1, denominators, convergents, k = 0,
      pk, qk, nk, sk, flag = True, P = {}, Q = {}},
   denominators = {c}, convergents = {c},
   While[flag, k = k + 1; sk = (a - omega*b)/(omega*d - c);
      nk = Floor[sk]; pk = a + nk*c; a = c; c = pk; (*advance p, q *)
      qk = b + nk*d; b = d; d = qk;
      denominators = Append[denominators, nk];      (*update lists *)
      convergents = Append[convergents, pk/qk];
      P = Append[P, pk]; Q = Append[Q, qk];
      If[nk == sk || k == max, flag = False]        (*check if done*)
      ];                                            (*end of loop  *)
   {denominators, convergents, P, Q }]              (*return output*)

Qlist[n_, m_] := Block[{ Cs = Rcf[Sqrt[n], m], Qs = { }, j, Q, s },
   Do[If[Mod[j, 2] == 1, s = Cs[[3, j]]^2 - n*Cs[[4, j]]^2;
      If[Floor[Sqrt[s]]^2 == s,
         Qs = Append[Qs, {j, Cs[[3, j]], s}]]], {j, 1, m}];
   Qs];
```

To factor the integer 91, for example, we use the command `Rcf[Sqrt[91],5]` which outputs four lists: the partial denominators, the convergents, the numerators, and the denominators of the convergents:

$$\{\{9,1,1,5,1,5\},\left\{9,10,\frac{19}{2},\frac{105}{11},\frac{124}{13},\frac{725}{76}\right\},\{10,19,105,124,725\},\{1,2,11,13,76\}\}.$$

The command `Qlist[91,5]` outputs its lists as $\{\{1,\, p_1,\, Q_1\},\, \{5,\, p_5,\, Q_5\}\}$:

$$\{\{1,\ 10,\ 9\},\ \{5,\ 725,\ 9\}\}.$$

That is, Q_1 and Q_5 are squares. Taking $Q_1 = 3^2$ and $p_1 = 10$ means that either $p_1 - 3 = 7$ or $p_1 + 3 = 13$ should have a factor in common with 91. Indeed they both do, as $7 \cdot 13 = 91$. ♢

Code 22: Window radii for the cicada. Adapting the following code produces a table much like that of Table IX.2. The list `W0` contains various relative angular velocities for the Moon's phases in moons/year; `P1` is a list of the denominators from `W0`. The function `moon[n]` outputs a list of the number of moons, to the nearest integer, which occur in the years from year 1 to year `P1[[n]]`, the year corresponding to item `n` in the list `P1`. With a full moon at time `j=0`, the function `see[n]`, computes the difference in days between year `j` and the full moon date nearest to year `j`, for all `j` up to year `P1[[n]]`. The function `nice[s]` will round a real number to the nearest tenth.

```
W0 = {12 + 2/3, 12 + 3/5, 12 + 4/7, 12 + 5/9, 12 + 6/11, 12 + 7/13,
     12 + 8/15, 12 + 9/17, 12 + 10/19};
P1 = {3, 5, 7, 9, 11, 13, 15, 17, 19};
moon[n_] := Round[Table[ j * W0[[n]], {j, 1, P1[[n]]}] ];
see[n_] := Table[ Abs[moon[n][[j]]/W0[[n]]-j]*365.25, {j,1,P1[[n]]}];
nice[s_] := Round[10*s]/10.;

Table[nice[see[j]], {j, 1, 5}]
```

```
 9.6   9.6    0.
11.6   5.8    5.8   11.6    0.
12.5   4.2    8.3    8.3    4.2   12.5    0.
12.9   3.2    9.7    6.5    6.5    9.7    3.2   12.9    0.
13.2   2.6   10.6    5.3    7.9    7.9    5.3   10.6    2.6   13.2    0.
```

Executing the last command outputs a table equivalent to the one above. ♢

Code 23: Applying the litmus test of Proposition X.3. The following code will generate a list of prospective years at which a transit of Venus may occur. The Mathematica command `Inverse[M]` returns the inverse matrix of the square matrix `M`. We use `U` for ω and `L0` for λ. The functions `V, EE,` and `W` respectively give the three dimensional positions of Venus and Earth and the projection of Venus from Earth upon the screen of the Sun.

```
xi=3.39*Pi/180;       (*tilt of V's orbital plane with respect to E*)
U = 1/(224.70/365.26);
L0 = 1/U^(2/3);                        (*distance of V from Sun, 0.723 AU*)
```

$$\texttt{V[t\_] := L0} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & \texttt{Cos[xi]} & -\texttt{Sin[xi]} \\ 0 & \texttt{Sin[xi]} & \texttt{Cos[xi]} \end{bmatrix} .\{\texttt{Cos[2*Pi*U*t],Sin[2*Pi*U*t],0}\};$$

```
                                                     (*V position *)
EE[t_] := {Cos[2*Pi*t],Sin[2*Pi*t],0}; (*Earth position at time t *)
                        ┌ Cos[2*Pi*t]  Sin[2*Pi*t] 0 0 ┐
          ┌0 1 0 0┐     │ -Sin[2*Pi*t] Cos[2*Pi*t] 0 0 │
W[t_] :=  └0 0 1 0┘  .  │      0            0      1 0 │ .Inverse
                        └      0            0      0 1 ┘
   ┌      1            0       0     Cos[2*Pi*t]-L0*Cos[2*Pi*U*t]      ┐
   │      0            1       0 Sin[2*Pi*t]-L0*Cos[xi]*Sin[2*Pi*U*t]  │
   │      0            0       1        L0*Sin[xi]*Sin[2*Pi*U*t]       │
   └ Cos[2*Pi*t] Sin[2*Pi*t]   0                   0                   ┘
   .{Cos[2*Pi*t],Sin[2*Pi*t],0,0};      (*end of definition of W[t] *)

norm[X_] := Sqrt[ X.X ];
SameSide[t_]: = (V[t].{Cos[2*Pi*t], Sin[2*Pi*t] ,0})
   *(EE[t].{Cos[2*Pi*t], Sin[2*Pi*t], 0}) > 0; (*a Boolean function*)
                              (*Are EE and V on same side of screen?*)
Near[t_] := (SameSide[t] && norm[W[t]]) < 0.05;(*a Boolean function*)
                                              (*Is V near a transit?*)
FindTransit[m_,n_] := Block[{date = {}, i},
    Do[If[Near[i], date = Append[date, i]];
       If[Near[i + 0.5], date = Append[date, i + 0.5]], {i, m, n}];
    date];
```

Calling the function `FindTransit[0,1000]` generates the list of possible transit years $\{0, 113.5, 227, 340.5, 348.5, 454, 462, 575.5, 689, 802.5, 916\}$. ◊

Code 24: A vector calculus approach to eclipse dates. The following code generates a list of eclipse dates in accordance with Proposition XII.5.

```
xi = 5.14*Pi/180; omega1 = 1.085196; s1 = 149.6*10^6; m1 = 384000;
test[n_] := Abs[ s1*m1*Sin[xi]*Sin[2*Pi*omega1*n]/   (*Boolean func*)
   (s1 - m1*Sqrt[1 - (Sin[xi]*Sin[2*Pi*omega1*n])^2])] < 9900;
Block[{dates = { }, j},
   Do[If[test[j], dates = Append[dates, j]], {j, 0, 250}];
   dates]
```

Executing the last statement produces eclipse dates: $\{0, 6, 12, \ldots, 247\}$. ◊

Code 25: A number theory approach to eclipse dates. The following code generates a list of eclipse dates in accordance with Lemma XII.8.

```
test2[ k_, mu_, eta_ ] :=                        (*a Boolean function*)
   Block[{T}, T = Floor[(mu + 38*k)/223];
      (T >= Ceiling[(-mu + 38*k)/223]
         && T <= Floor[(eta + 61*k)/358]
         && T >= Ceiling[(-eta + 61*k)/358])];
eDates[mu_, eta_] := Block[{dates = { }, j},
   Do[If[ test2[j, mu, eta], dates = Append[dates, j] ], {j, 0, 250}];
   dates]
```

The function call `eDates[21, 33]` produces eclipse dates: $\{0, 6, 12, \ldots, 247\}$. ◊