Retirement Income Analysis
with scenario matrices

William F. Sharpe

12. Incomes and Fees

Scenarios

Our key matrices have two dimensions. Each row contains a *scenario* – a possible future history of longevity, income received and fees paid. We generate a large number of equally likely future scenarios (typically 100,000), in order to try to understand the range of possibilities associated with a given retirement income strategy or set of strategies. The goal is to obtain the "best" set of such possibilities, but of course it is difficult (if not impossible) to specify a measure (utility) that can be used to assess the desirability of a particular strategy, let alone find the best possible one from a very large set of possibilities.

It would be helpful if recipients such as Bob and Sue could look at each of 100,000 scenarios for one strategy, then each of 100,000 scenarios for another, choose the preferred one, compare it with the 100,000 scenarios for yet another, choose the preferred one and so on until the very best strategy is determined. We have called this the "optometrist approach". But of course it is not feasible. Even the first step is likely to be too much. Actual human beings may be able to process information about a few scenarios, maybe a dozen or even a hundred. And it may be easy to dismiss a strategy out of hand for failing to provide minimal desires. But while examining selected scenarios one-by-one is likely to be helpful, this should be supplemented with other types of analysis.

Most of the analyses introduced in this and the next chapter summarize scenario matrix information column-by-column rather than row-by-row, thereby reducing the dimensionality of the results (for example, from 100,000 curves to 50). But it is nonetheless useful to start the analysis of a strategy by looking at a manageable number of possible scenarios. This section shows how this can be done.

First we need to add to the *analysis_create* function, a set of elements to provide information for the *plotScenarios* analysis. Here are the required statements.

```
% plot scenarios
   analysis.plotScenarios = 'n';
 % plot scenarios: set of cases with real (r) or nominal (n) and
 %   income (i), estate (e) and/or fees (f)
   analysis.plotScenariosTypes = { 'ri'  'rie'  'rif'  'rief' };
  % plot scenarios: number of scenarios
   analysis.plotScenariosNumber = 10;
```

The first element determines whether or not scenarios should be plotted. The next indicates which aspects are to be plotted. For this we use a *cell array,* indicated by curly brackets { and }. In this case, the members of the array are strings, but a cell array can contain strings, numbers, vectors, and/or other types of variables. Here, each string indicates the components to be used for a separate scenario graph.

The first letter in each string indicates whether real *(r)* or nominal *(n)* values are to be shown. In most cases, real values are preferred, since our pricing kernel is based on real values and we assume that retirees are concerned with the amounts of goods and services their incomes can procure. However, in some cases it may be useful to show nominal values in one plot, then real values in another, to indicate the danger of focusing on the former rather than the latter. The remaining letter or letters in a string indicate the aspects to be plotted: *(i)* income for recipients while they are alive, *(e)* money paid to the estate after the last recipient dies and *(f)* fees paid to financial firms and advisors.

The final element indicates the number of scenarios to be plotted in each graph. The default is 10, which is not likely to try the patience of a viewer, but this element could be set to any desired value up to the number of scenarios that have been generated (typically 100,000).

As with other elements for the analysis data structure, any of these can be changed after the structure has been created but before the *analysis_process* function is called.

The next task is to add statements to produce the graph(s) to the *analysis_process* function. For convenience, we use one external function (*analPlotScenarios*) for each requested case, providing this function with one of the strings taken from the corresponding cell array. Otherwise we follow the usual approach, creating a figure, calling the external function to produce the information, then processing the result. Here are the statements added to the *analysis_process* function:

```
% analysis: plot scenarios
    if analysis.plotScenarios == 'y'
      % find types
        types = analysis.plotScenariosTypes;
      % create figures
        for i = 1:length( types )
          % create figure
            createFigure( analysis, client );
          % call external function analPlotScenarios
            analPlotScenarios( analysis, client, market, types{i} );
          % process figure
            analysis = processFigure( analysis );
        end;
    end;
```

The external function that creates the desired plots is rather complex since it uses sampling, animation and provides for different combinations of information. For completeness we show the entire program in the next three pages, then describe key features, choosing not to dwell on details that only a veteran Matlab programmer (such as the author) could love.

```
function analPlotScenarios( analysis, client, market,  plottype );
  % plot scenarios for income, estates and/or fees
  % called by analysis_process function

  % make plottype lower case
    plottype = lower( plottype );

  % add labels
    set( gcf, 'name', ['Scenarios: '  plottype] );
    set( gcf, 'Position', analysis.figPosition );
    grid on;
    title([ 'Scenarios'], 'color',[ 0 0 1] );
    xlabel( 'Year' );
    if findstr(plottype,'r') > 0
      ylabel( 'Real Income, Estate or Fees' );
    else
      ylabel( 'Nominal Income, Estate or Fees' );
    end;
    hold on;

 % set colors for states 0,1,2,3,4 and fees (5)
    % orange; red; blue; green; orange; black
    cmap = [ 1 .5 0 ; 1 0 0; 0 0 1; 0 .8 0; 1 .5 0; 0 0 0 ];


 % convert client income and fees to nominal values if required
    if findstr( plottype, 'n' ) >  0
       client.incomesM = market.cumCsM .* client.incomesM;
       client.feesM    = market.cumCsM .* client.feesM;
    end;

 % extract sample matrices for at least 100 scenarios
    n = max( 100, analysis.plotScenariosNumber );
    [nscen nyrs] = size( client.incomesM );
    firstScen = randi( [1 nscen – n] );
    lastScen = firstScen + n-1;
    scenPStates = client.pStatesM( firstScen:lastScen, : );
    scenIncomes = client.incomesM( firstScen:lastScen, : );
    scenFees    = client.feesM( firstScen: lastScen, :);

 % set personal states to be shown
    if findstr( plottype, 'I' ) > 0
      states = [1 2 3];
    end;
    if findstr( plottype, 'e' ) > 0
      states = [states 4];
    end;
```

```
% find maximum value for y axis
    incomeCells = zeros( size( scenPStates ) );
    for i = 1:length( states )
       incomeCells = incomeCells + ( scenPStates == states(i) );
    end;
    maxIncome = max(max( ( incomeCells>0).*scenIncomes ) );
% if fee is to be included, find maximum fee for sample states
   if findstr( plottype, 'f'  ) >0
      maxFee = max( max(scenFees) );
   else
      maxFee = 0;
   end;
% set maximum for y axis
   maxY = 1.01*max( maxIncome, maxFee);

% set axes
   axis([ 0 nyrs 0 maxY ]);

% set shade and delay parameter
   shade = analysis.animationShadowShade;
   delays = analysis.animationDelays;
   delayChange = ( delays(2)-delays(1)) / (analysis.plotScenariosNumber -1) ;

% show scenarios
   delay = delays(1);
   for scenNum = 1 : analysis.plotScenariosNumber

 % plot incomes
     incomes = scenIncomes( scenNum, : );
     pstates = scenPStates( scenNum, : );
     for pstate = states
           x = find( pstates == pstate );
           if length(x)  >  0
              y =incomes(x);
              plot( x, y, '-*', 'color', cmap(pstate+1,:), 'Linewidth', 2.5 );
           end;
    end;

 % plot fees
      if findstr( plottype, 'f'  )> 0
          fees = scenFees( scenNum, : );
           plot( 1:nyrs, fees, '*',  'color', cmap(6,:), 'Linewidth', 2.5 );
      end;
```

```
  % pause
      pause( delay );
      delay = delay + delayChange;    % re-plot incomes using shading
      for pstate = states
        x = find(pstates == pstate);
        if length(x)  >  0
          y = incomes( x );
          clr = shade * cmap(pstate+1, :) + (1-shade)*[1 1 1];
          plot( x, y ,'-*', 'color', clr, 'Linewidth', 2.5 );
        end;
      end;

  % re-plot fees using shading
      if findstr( plottype, 'f'  )  > 0
         clr = shade*cmap(6) + (1-shade)*[1 1 1];
         plot( 1:nyrs, fees, '*',  'color', clr, 'Linewidth', 2.5 );
      end;

   end; % for scenNum = 1:analysis.plotScenariosNumber

end % plotScenarios(analysis, client,market, caseNum;
```

The first section sets up the figure, then adds labels, using the appropriate label for the y-axis depending on whether real or nominal values are to be shown. The next provides the colors to be used. As in the survival probabilities graph, we use red when person 1 is alive, blue when person 2 is alive and green when both are alive. In addition, we use orange to indicate money paid to the estate and black to represent fees paid. The imagery of the latter is intentional, since paying some (but not all) fees can be likened to money being lost in a (space/time) black hole.

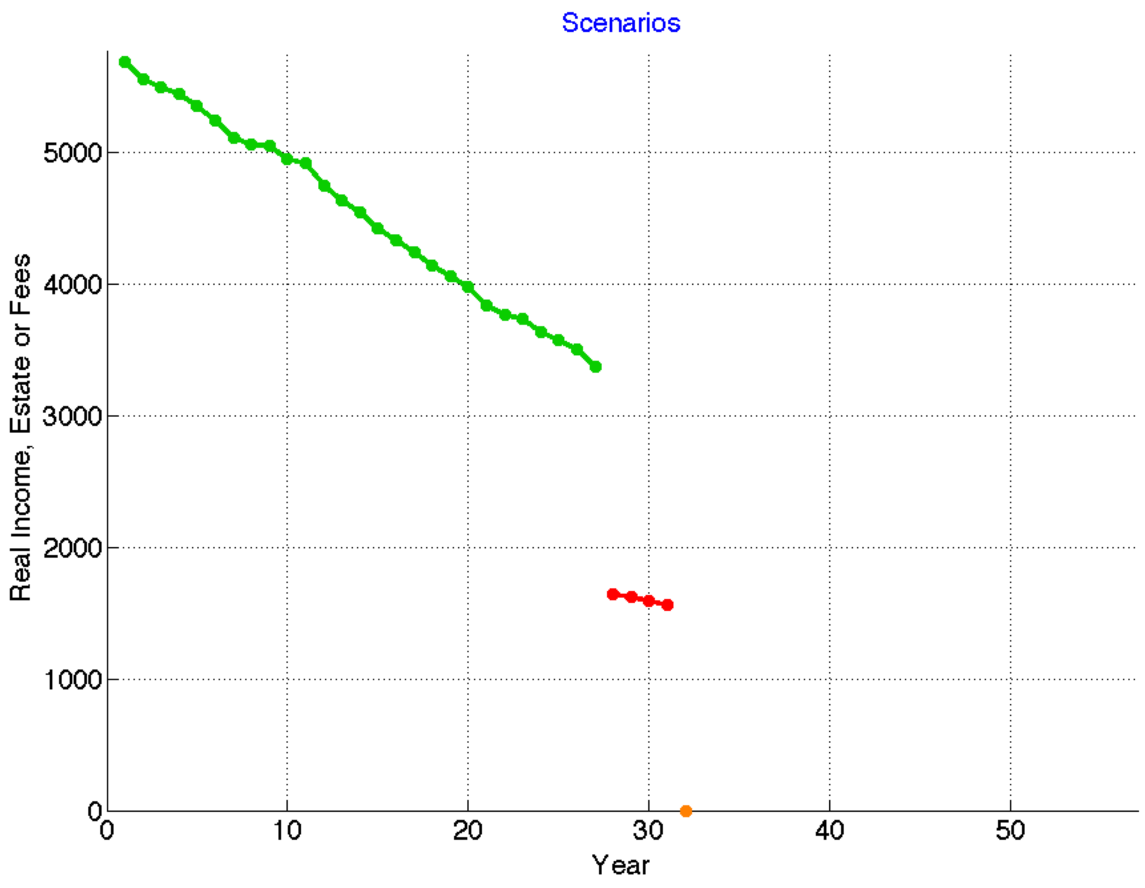
The next section changes the *client.incomesM* and *client.feesM* to nominal values, if required. This may seem dangerous but is not, since it only affects the copy of the client data structure used within this function.

The next section extracts a set of rows from which scenarios will be drawn, one by one, for display. There will be at least 100 of these and more if required to show the requested number of scenarios. The following sets of statements find and set the maximum value for income and possibly fees to be shown on the y-axis, based on the set of sample scenarios.

The remaining statements create the plots. First, the elements in the analysis data structure controlling the shade of shadows and the first and last delay times are used to set internal variables. Then the scenarios are shown, one at a time. The incomes and estate payments (if requested) for a scenario are plotted, state by state, then the fees (if requested). Following a delay of the desired length, all the information is re-plotted, using the shade specified in the analysis data structure. This continues until the desired number of scenarios have been shown.

As is often the case, a picture is worth more than a thousand words (and, a fortiori, many lines of program code). So let's turn to some examples.

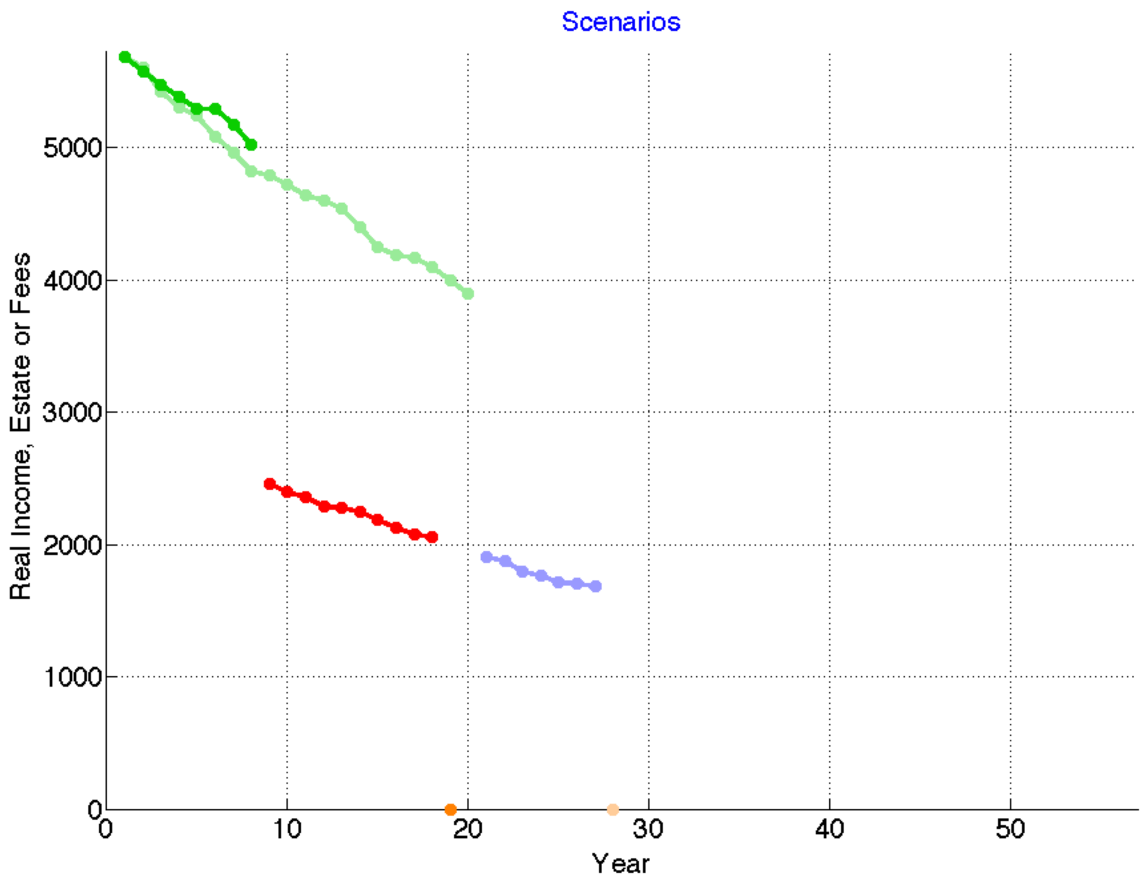
Here is a case showing one scenario (*analysis.plotScenariosNumber = 1*), using real values of incomes and estate payments (*analysis.plotScenariosTypes= {'rie' }*) for our fixed annuity with constant nominal payments. For emphasis, we have set *analysis.animationShade = 1.*



In this case, both Bob and Sue live for 27 years, then Bob dies, leaving Sue with half the nominal income. She lives another 4 years, then dies leaving no estate (zero).

As expected, real income declines as inflation takes its toll on the purchasing power of the constant nominal income. The rate of decline is not constant, of course, but relatively close to it, since we have assumed a relatively small standard deviation of inflation.

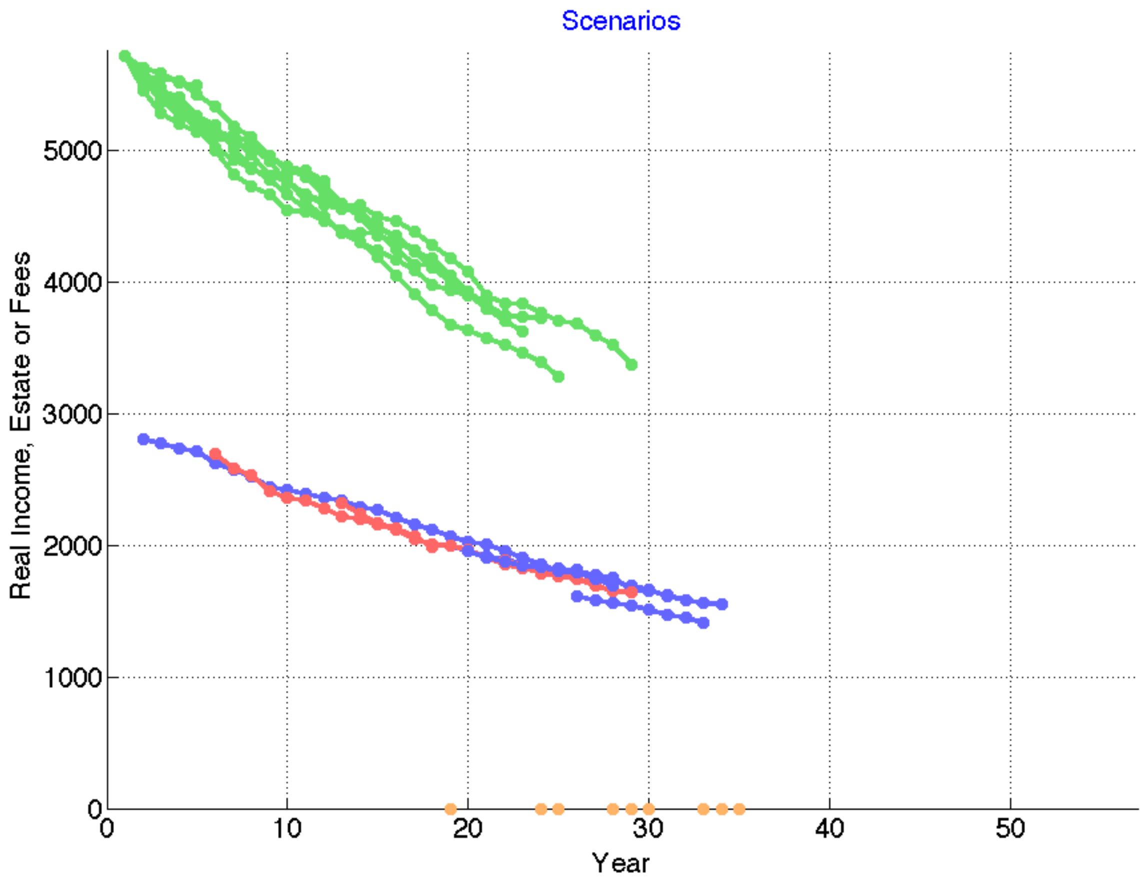
Now let's set the *analysis.animationShadowShade* back to its standard value of 0.4, request more scenarios, and stop just after the second has been drawn. Here is a sample result:



Note that in the first (lighter) scenario, Sue (blue) survived after their relatively long life together. But in the second (darker) scenario, Sue died rather early (after 8 years), with Bob living another 10 years. Of course in both cases the estate was zero, as intended (leaving nothing for any children or charities).

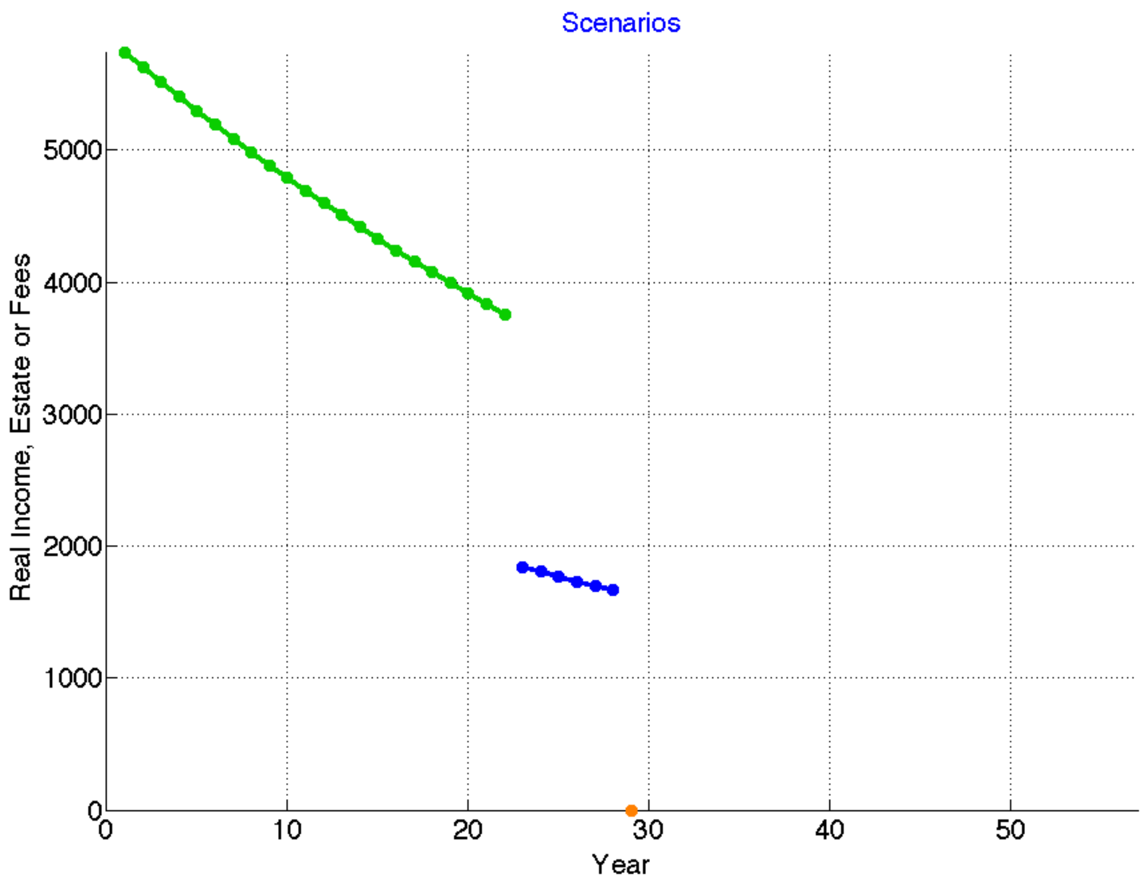
The figure shows that there is some variation in income across years when both are alive in every scenario, since inflation is uncertain to some degree. But the largest source of uncertainty here is clearly longevity.

The next figure, which includes ten scenarios, gives a better idea of the possible variations across scenarios:

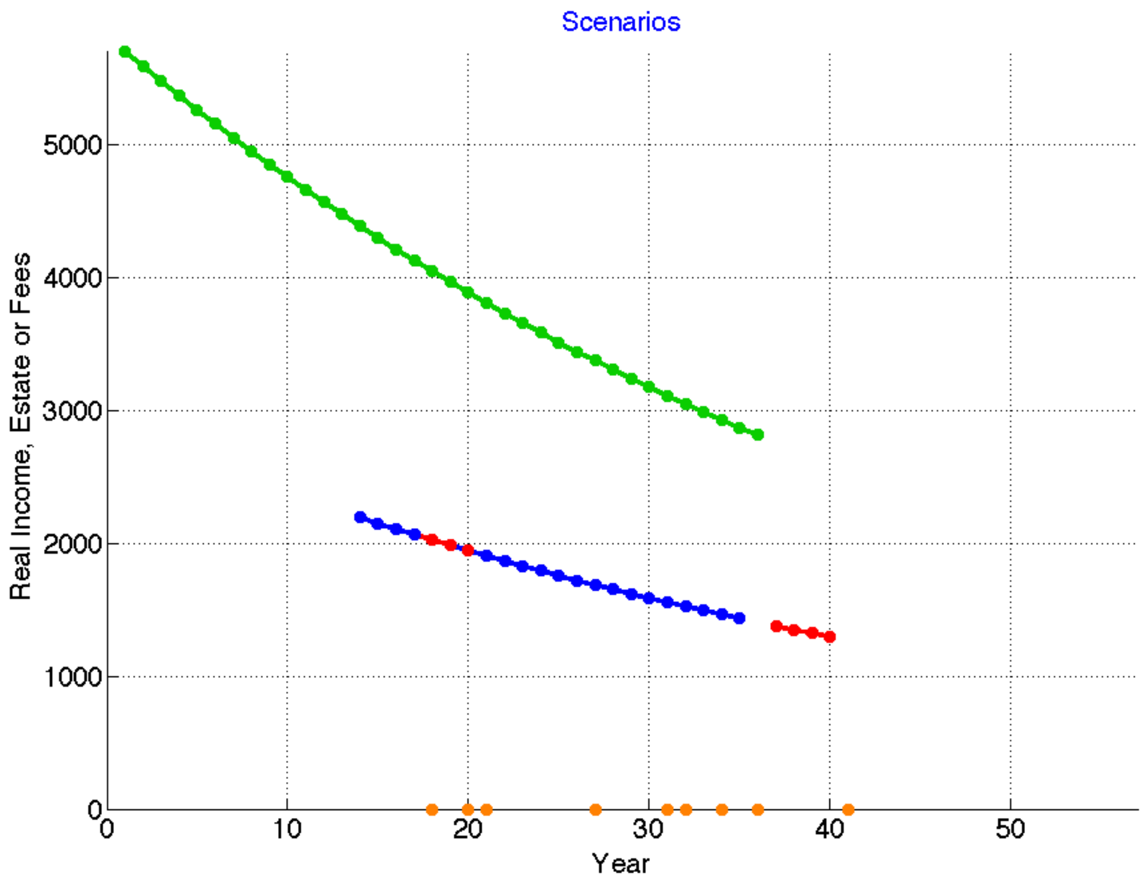


Here the effect of inflation uncertainty is more visible. Compare, for example, the real incomes in year 20 for the scenarios in which both Bob and Sue are both still alive. The amounts range from roughly $3,600 to $4,100. While nominal incomes follow the same path in each scenario, real incomes do not.

To emphasize the point, here is one scenario for a fixed immediate real annuity with graduated payments for which each year's real payment is 0.98 times that of the prior year. As intended, the year-to-year percentage chances in real income are the same for each year as long as the personal state remains the same.

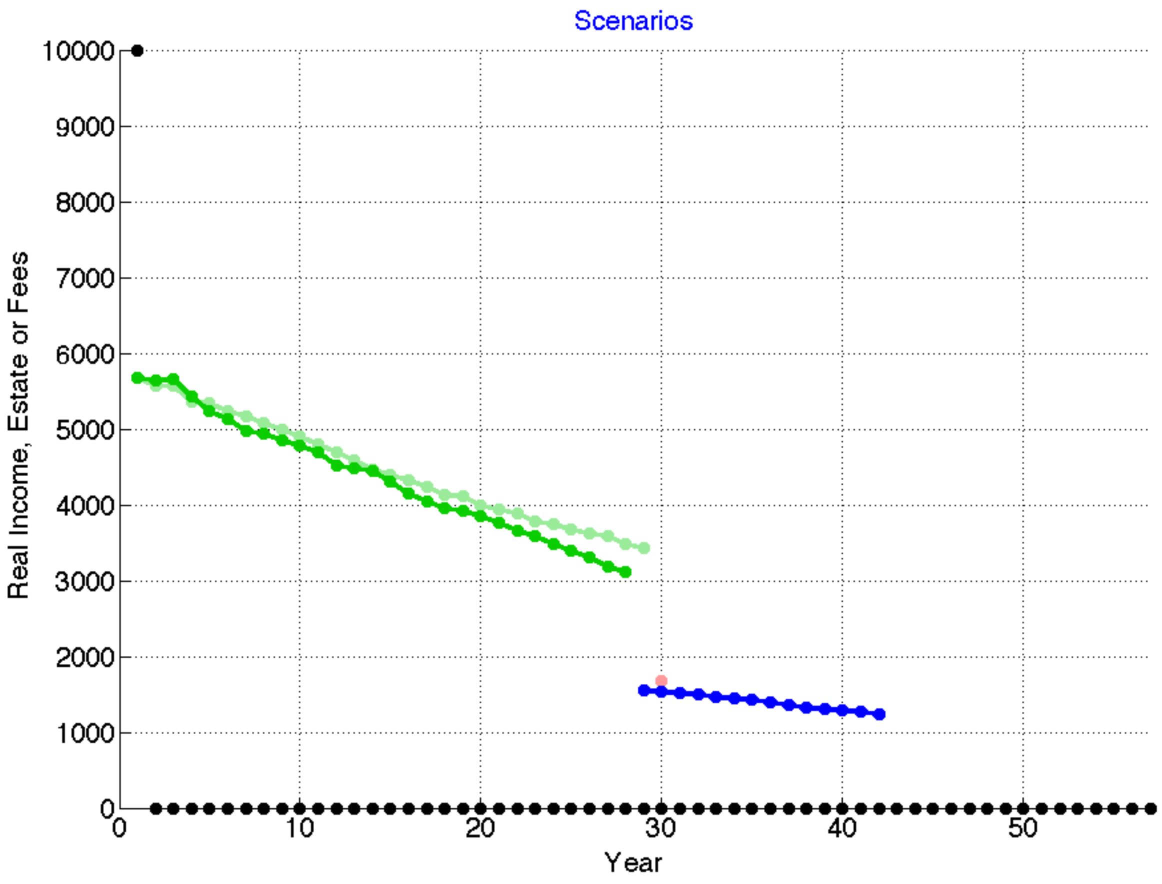


Not surprisingly, this graduated payment fixed real annuity provides the same results across scenarios in each year for a given personal state. Here is a figure showing ten scenarios.



The only uncertainty in this case is that associated with longevity. In any year in which both Bob and Sue are alive, their income will plot on the green curve. In any year one of them is alive, income will be that shown on the curve with blue and red dots (some of which cover up prior ones). And no matter what happens, there will be no estate, as intended.

We illustrate one last scenario analysis before moving on to other matters. If *analysis.plotScenariosTypes{ }* includes *'rief'* , fees will be included in the figure, and the scale chosen as needed to include all the information. Here are two scenarios for our fixed annuity with constant nominal payments.



The bad news is the high fee at the outset. The good news is that there are no fees thereafter – each is zero. This is not so for other retirement income strategies. As we will see, many strategies that include non-annuity sources of income involve the payment of fees in future years. Since such fees can seriously diminish retirees' spendable incomes, it is important to take them into account when evaluating alternative approaches. We of course state all incomes on a net-of-fee basis. But it is sometimes useful to examine fees as well, and to measure their impact. We will do so again in the next chapter.

This concludes our discussion of scenario-by-scenario plots for a subset of scenarios (rows in a scenario matrix). We turn now to approaches that attempt to summarize all the information in such matrices.

Annual Income Distributions

The previous section showed how one might look at the information in a scenario income matrix one row (scenario) at a time. We now consider ways to look at it one column (year) at a time. To illustrate, we focus on the real income obtained in year 20 for the case in Chapter 10 in which the Smiths invested $100,000 in a fixed annuity with constant nominal payments. Since the amount received would differ if both were alive (personal state 3) or if only one were (personal state 1 or 2), our example will include only scenarios in which both are alive. In this case there were 39,504 such scenarios out of the total of 100,000 scenarios in year 20.

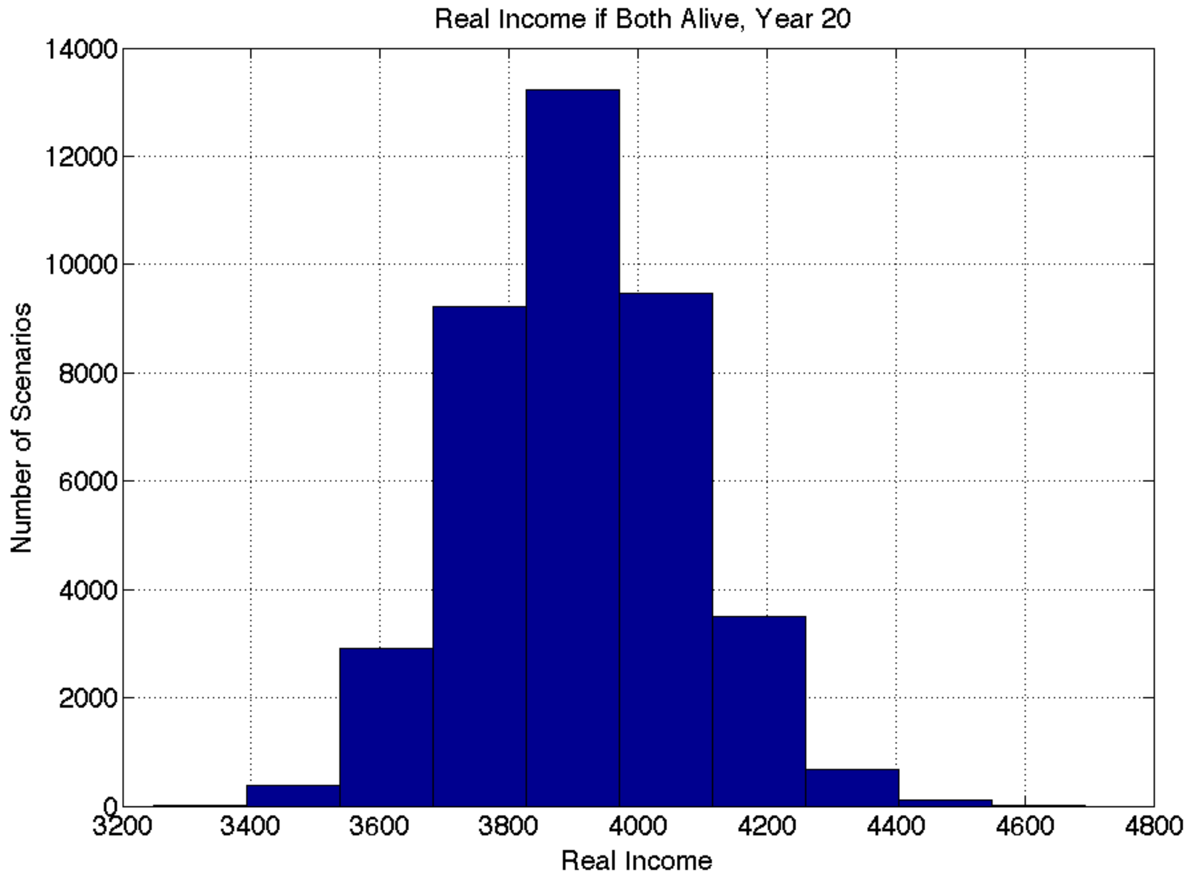
We begin by extracting the incomes for the year and the scenarios in which the personal state equals 3:

```
yr = 20;
ps = client.pStatesM( :, yr );
incs = client.incomesM( :, yr );
ii = find( ps == 3 );
y = incs(ii);
```

The simplest possible way to portray the distribution of these incomes is to use the Matlab function for a histogram:

hist(y)

Adding grid lines and labels produces the following figure.



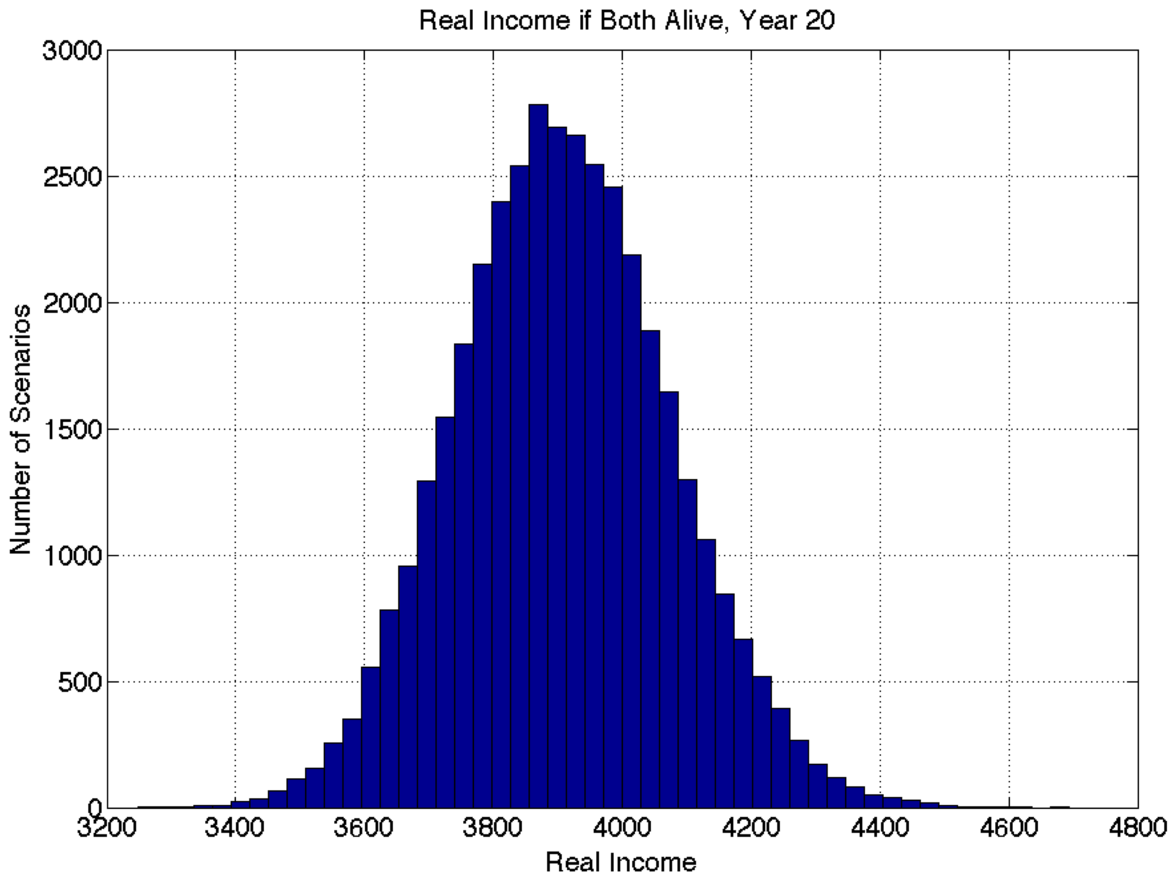
The function groups the observations in a series of consecutive bins, then shows how many fall in each one. The default is for ten bins, as shown here.

While useful, this summarizes the information rather crudely. The range of possible incomes is clear (from $3,400 to $4,700 per year) . And incomes in more scenarios fall in the bin in middle of the distribution than in any other. But there are clearly better ways to portray the information.

It is a simple matter to ask for more bins. For example:

```
hist( y, 50 )
```

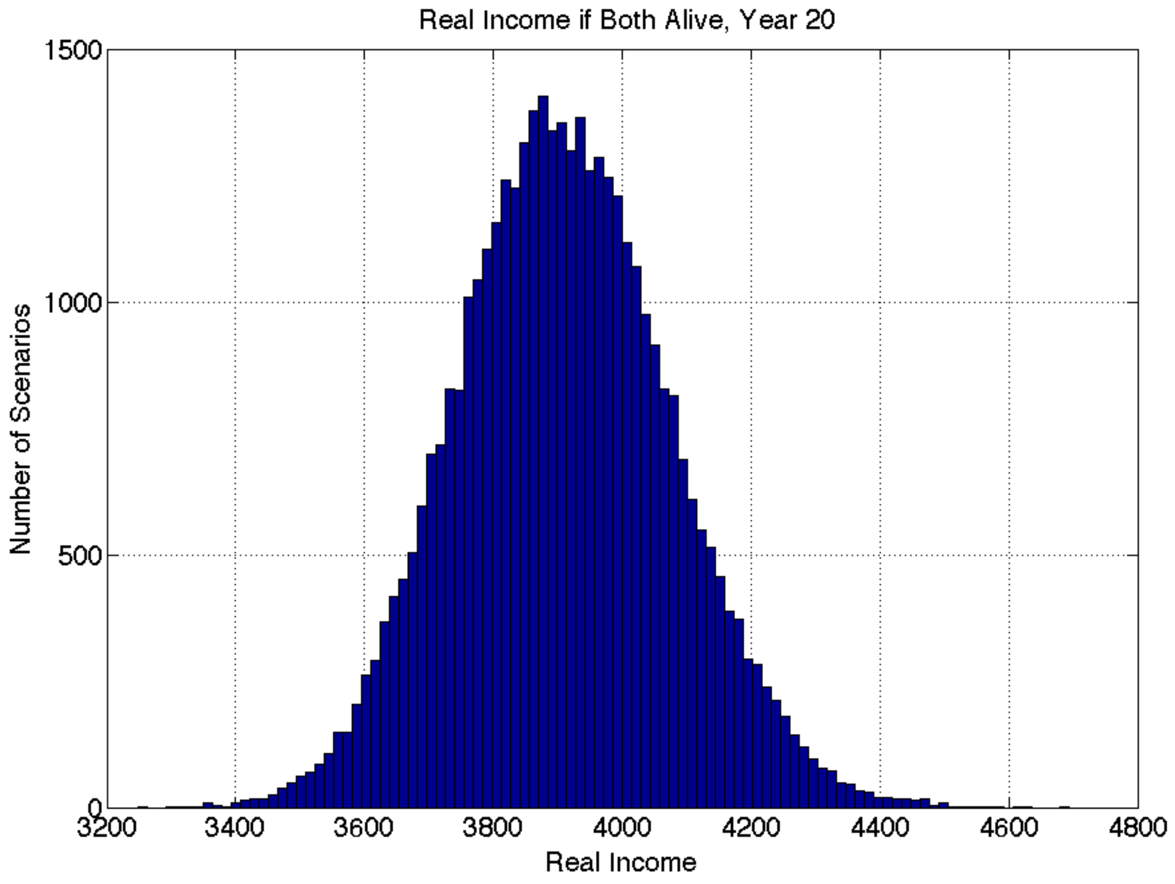
produced the following figure with incomes grouped into 50 bins:



Further,

```
hist( y, 100 )
```

resulted in the following:

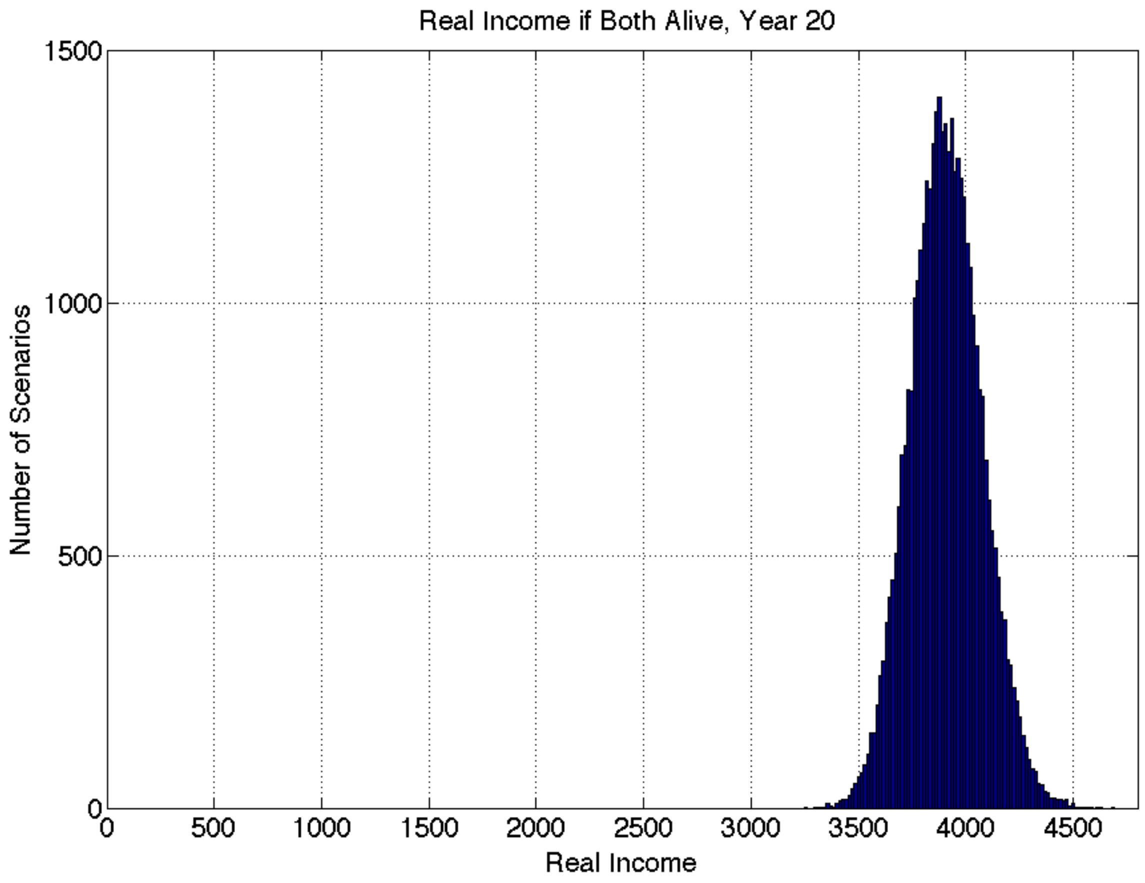


We could go on, calling for more bins. But the characteristics of the income distribution seem reasonably clear at this point. The most likely outcomes are in those near $3,900 per year, but the actual real income one might receive could be up to $500 less or $600 or so more.

The numbers on the axes provide an important part of this story. The range of possible outcomes is not huge when viewed in *relative* terms. For example, $4,400 is only 29% greater than $3,400. But this is not obvious from a cursory look at the figure. To better provide a view of the scale, it is useful to start the horizontal axis at zero. This is is easily done:

```
        ax= axis;  ax(1) = 0; axis(ax);
```

The result, shown in the next figure, provides needed perspective, which may be worth any associated loss in detail.



While histograms are both useful and familiar, they have serious drawbacks. First, it can be difficult and time-consuming to find a level of detail (e.g. number of bins) that will usefully portray the distribution of possible outcomes; generally the choice is somewhat arbitrary. Second, some information will inevitably be lost, since within a bin no account is taken of the distribution of outcomes. For these reasons, we choose a different way to portray possible outcomes.

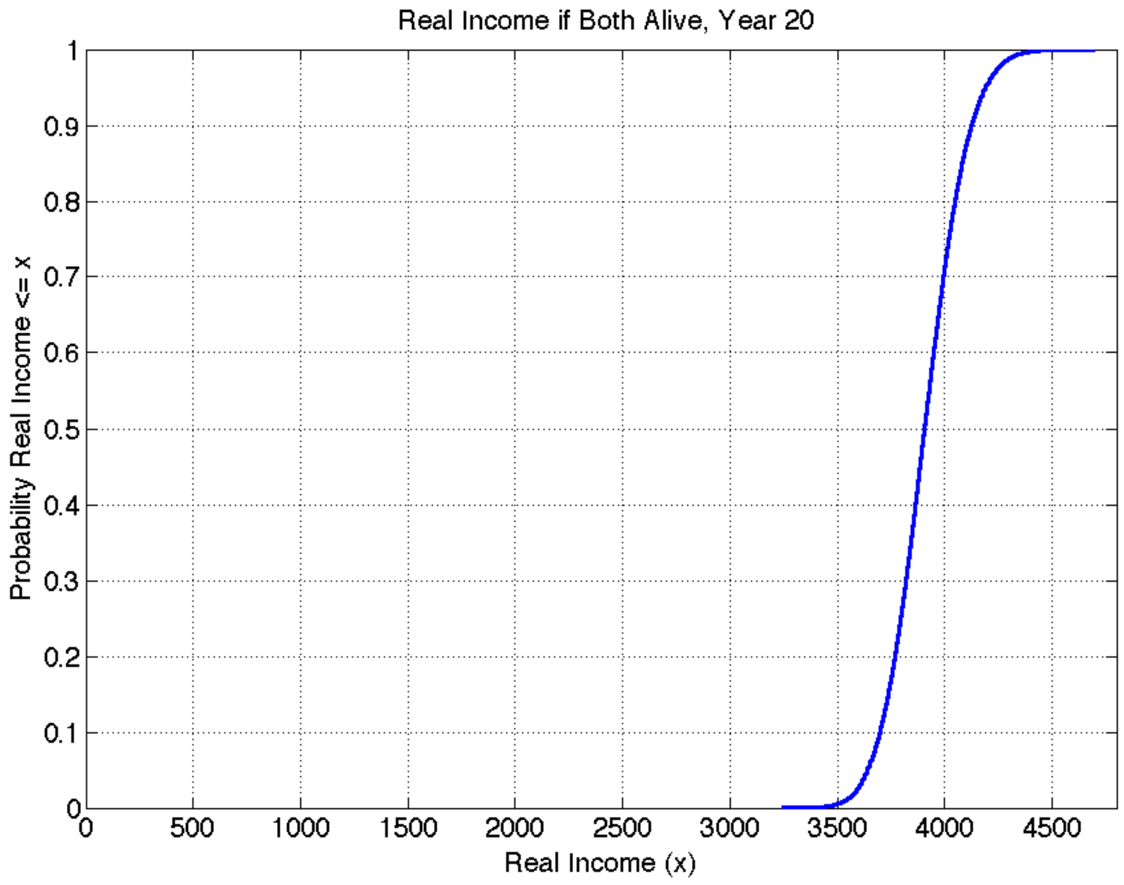
Here is the Wikipedia entry for a *cumulative distribution function*:

> *In probability theory and statistics, the* ***cumulative distribution function (CDF)****, or just* ***distribution function****, evaluated at 'x', is the probability that a real-valued random variable X will take a value less than or equal to x. In other words, CDF(x) = Pr(X<=x), where Pr denotes probability.*

This is easily done in Matlab since each of the possible incomes in our vector y is equally likely . We need only three statements:

```
yx = 1 : 1:length( y );
yx = yx / length(yx);
plot (sort ( y, 'descend' ) , sort( yx, 'descend' ) );
```

After adding labels, a title and grid lines, we get this graph:



One can think about this as a graph with many points, each indicating a possible real income (on the horizontal axis), sorted in ascending order, with each plotted one position above the prior one. Adjacent points are connected with lines, but there are almost 40,000 of them, so the result appears to be a smooth continuous curve. In cases with few possible incomes, the connecting lines might more visible.

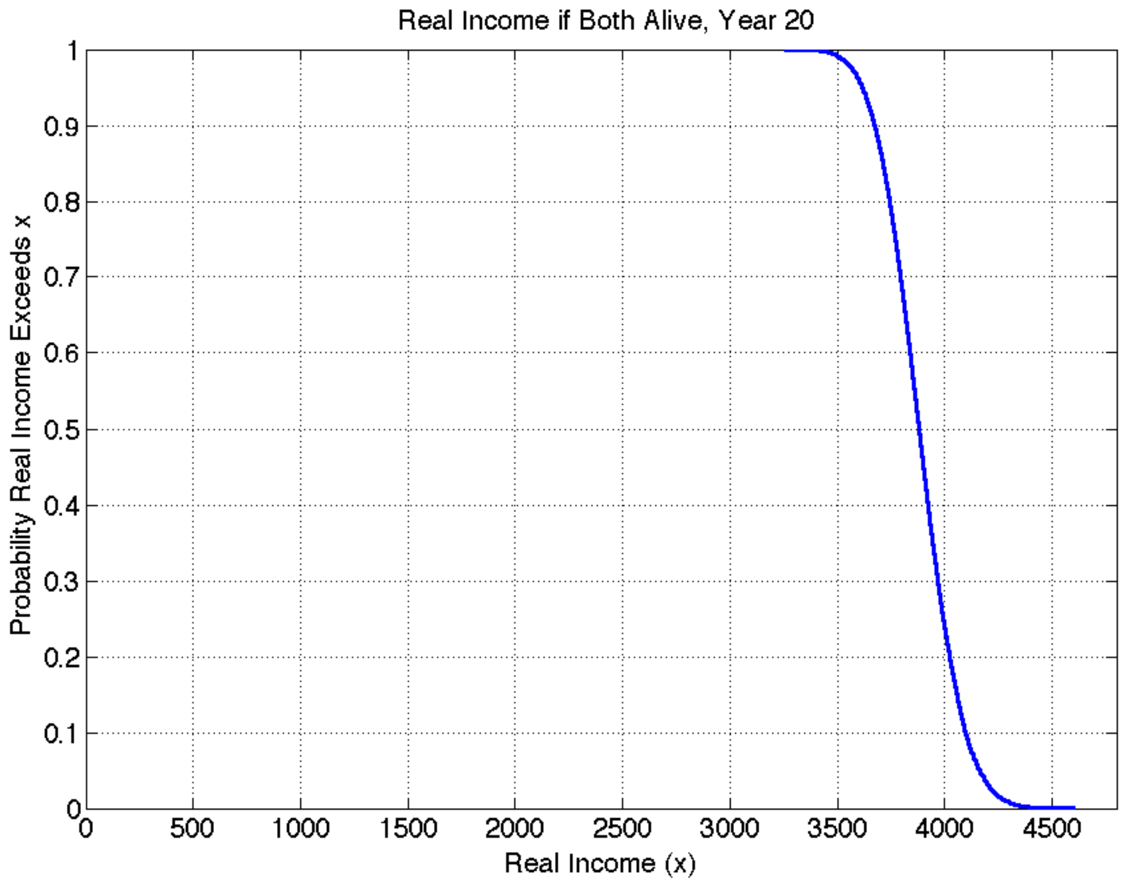
Note that this portrayal requires no decisions to be made about the number of bins, the range of incomes to be included in each one, etc.. All the information is included, with no arbitrary groupings.

However, such a portrayal has one drawback. Consider an alternative strategy with incomes that would plot on a curve that is everywhere to the right of the one shown. This would obviously be a superior strategy, since for any chosen income (x axis) there would be a smaller failure probability (y axis). Clearly, a shift to the right is good. But we could equally say that a shift down is good, since a smaller failure probability for any given income is preferred. In this and other conventional cumulative distribution plots, the x-axis plots a *good thing* (more income is better) while the y-axis plots a *bad thing* (a greater failure probability is worse). We prefer to use a slightly different approach, where each axis plots a *good thing,* making it desirable to go either up or to the right.

We need only alter the plot command to:

```
plot (sort (y,'ascend') , sort(yx,'descend') );
```

Here is the result.



Note that we have changed the y-axis label, stated it in less threatening terms (using "*exceeds*" instead of "*>*").

This approach should resonate with investors who ask “how bad could it be?”. Each point on the curve provides an answer to this question, though all are probabilistic except the one at the top of the diagram.

The figure also provides answers to other questions often posed probabilistically. The *x* value for *y=0.5* can be taken as the *median* of the distribution since there is close to a 50/50 chance that the actual income will equal or exceed it. The *x* value for *y=0.9* can be interpreted as the 10% *value-at-risk*, since there is a 90% chance that the income will be equal or greater than that amount, and hence a 10% chance that it will be worse.

We do not show some other commonly-used statistics such as the *expected income* or *standard deviation* of income.

The former is the probability-weighted mean of a set of possible future outcomes. Since each of our incomes is equally probable, the expected value would be the unweighted mean (average) value. Unfortunately, many people interpret mean values as if they were medians. For example consider this (actual) assertion, “the average water bill on the Monterey Peninsula was *x*, so 50% of the customers paid less than that.” This would be true only if the distribution were nicely symmetric around a central point (which the water bill distribution was not). Similarly, our distributions are not likely to be symmetric, since we assume lognormal returns and inflation rates. Moreover many methods for providing retirement income involve complex multi-year investment and spending policies which provide income distributions that are far from symmetric.

The standard deviation measure has similar drawbacks. If a variable's distribution conforms with the classic normal shape, close to two-thirds of the values will fall between (a) a value equal to the mean minus the standard deviation and (b) a value equal to the mean plus the standard deviation. But our returns and inflation values are not normally distributed, nor are retirement incomes likely to be, so either so this may not be the case. Fortunately, one can read the probability that income will fall in a given range directly from a cumulative distribution graph – it is simply the vertical distance between the corresponding end points.

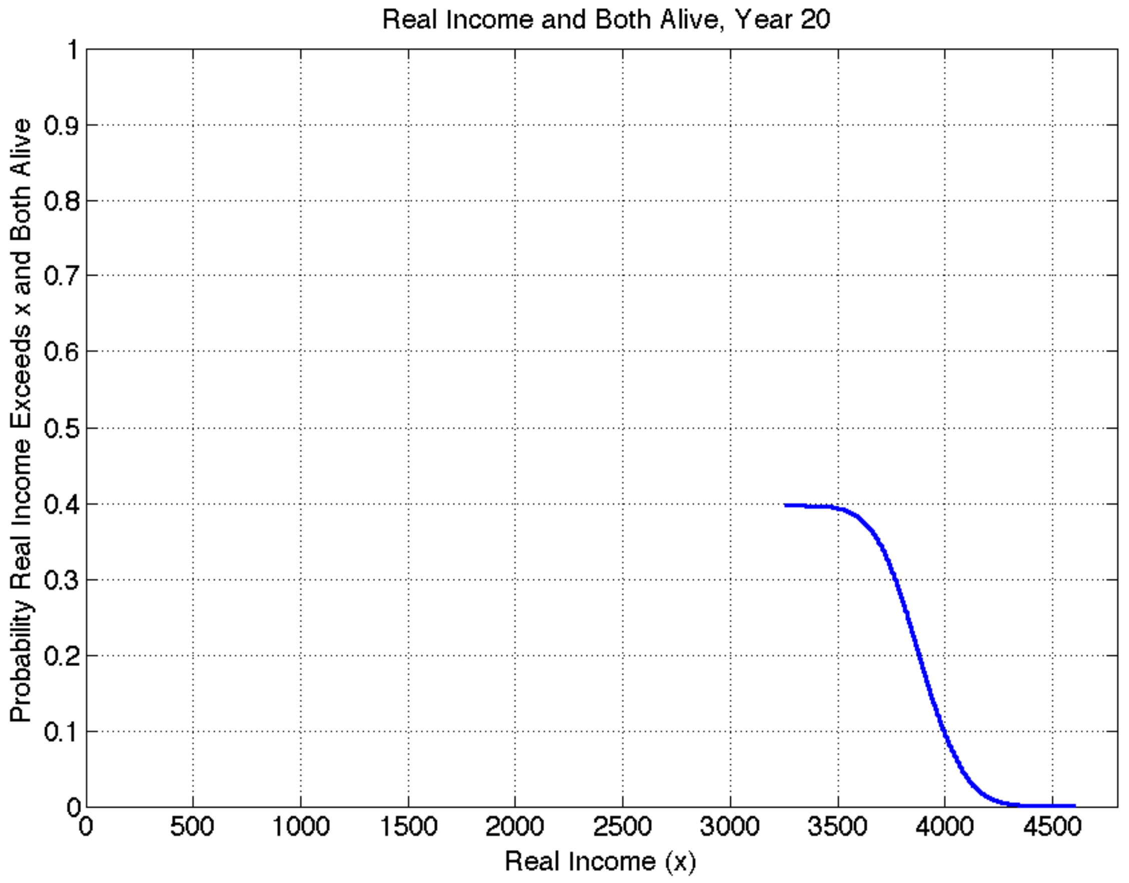
The bottom line is that a cumulative distribution can efficiently summarize the range of incomes that can be provided by a particular strategy or combination of strategies. It may not be the simplest possible approach, but to use a quotation sometimes attributed (possibly erroneously) to Albert Einstein: *everything should be as simple as possible, but not simpler.*

Before proceeding to implementations involving income in multiple future years, we need to consider one more issue. The prior graph shows income *if* both recipients are alive. In this sense it is *conditional*. But we know that in this case there only a roughly 40% chance that both Bob and Sue will be around to collect income twenty years hence. For perspective, it might be useful to portray this fact as well, showing joint probabilities – that is the probability that income equals or exceeds an amount *and* that the chosen personal state or states will take place.

This is easily done. We simply compute the probabilities by dividing each observation number by the total number of scenarios:

```
yx = 1: 1: length( y );
yxx = yx  /  size( client.incomesM, 1 );
plot( sort(y, 'ascend'), sort(yxx, 'descend') );
```

The resulting *unconditional* distribution is shown below. It provides additional perspective and could supplement, or supplant the *conditional* distribution shown earlier.



We are now ready to add the statements and programs needed to produce income distribution graphs.

We begin by adding the needed elements to the *analysis_create* function:

```
% plot income distributions
   analysis.plotIncomeDistributions = 'n';
 % plot income distributions: set of cases with real or nominal (r/n) and
 %   conditional or unconditional (c/u) types
   analysis.plotIncomeDistributionsTypes = { 'rc' 'ru' 'nc' 'nu' };
 % plot income distributions: sets of states (one set per graph)
   analysis.plotIncomeDistributionsStates = { [3]  [1 2] };
 % plot income distributions: minimum percent of scenarios
   analysis.plotIncomeDistributionsMinPctScenarios = 0.5;
 % proportion of incomes to be shown
  analysis.plotIncomeDistributionsProportionShown = 1.00;
 % plot income distributions: percent of maximum income plotted
   analysis.plotIncomeDistributionsPctMaxIncome = 100;
```

The first element indicates whether or not to produce any income distribution plots. The second provides the type for each desired graph. Here too we use a cell array of strings. In each one, the first letter indicates whether *(r)* real or *(n)* nominal values are to be shown, while the second letter indicates whether *(c)* a conditional or *(u)* an unconditional graph is desired. The next cell array includes one or more vectors, each of which indicates the states to be included when computing the information for a graph. Unless changed, these settings will produce eight graphs, four types for each of two sets of states.

The next element indicates the minimum percent of scenarios required for a distribution to be shown. This is included to avoid plotting results for years in which there are relatively few scenarios with incomes for the chosen states. Why? Because in such cases the number of simulated alternatives will be small, the results possibly unrepresentative and, most important, the plot ugly. The default value is 0.5%, which would require 500 scenarios for our standard case with 100,000 scenarios.

The final two elements are designed to allow for cases in which there are a few scenarios with extremely large incomes which, if included, would dwarf the display of the majority of the outcomes. In general, one should leave the proportion to be shown at 1.00 and the percent of maximum income to 100, observe the results, then select lower values if needed.

The next step is to include in the *analysis_process* function, statements that will generate calls to an external function *analPlotIncomeDistributions* to create the desired graphs. Here they are:

```
% analysis: plot income distributions
   if analysis.plotIncomeDistributions == 'y'
     % find states;
       states = analysis.plotIncomeDistributionsStates;
     % find types
       types = analysis.plotIncomeDistributionsTypes;
     % create figures
       for i = 1:length(types)
         for j = 1:length(states)
           % create Figure
            analysis = createFigure( analysis , client );
           % call external function analPlotSurvivalRates
             analPlotIncomeDistributions (analysis,  client,  market,  types{i},  states{j} );
           % process figure
            analysis = processFigure (analysis);
         end; %j
       end; %i
   end;
```

Nothing surprising here – the external function is called for each desired combination of *type* and *states,* generating a separate plot each time.

The *analPlotIncomeDistributions* function is complex, and we will only summarize its main features. Here it is in full.

```
function analPlotIncomeDistributions( analysis, client, market, plottype, states)
 % plots income distributions using personal states in vector states

 % initialize graph
   set( gcf, 'name', [ 'Income Distributions ' plottype ] );
   set( gcf, 'Position', analysis.figPosition );
   grid on;
 % make plottype lower case
   plottype = lower( plottype );
 % set colors for states 0,1,2,3 and 4
   % orange; red; blue; green; orange; black
   cmap = [ 1 .5 0 ; 1 0 0;  0 0 1;  0 .8 0; 1 .5 0 ];

 % set condition labels
   if findstr( 'c', plottype )>0
       condition = 'if ';
     else
       condition = 'and ';
   end;

  % set real or nominal text
   if findstr( 'n' ,plottype )> 0
       rntext = 'Nominal ';
     else
       rntext = 'Real ';
   end;

  % set states text
   statestext = [ condition 'States = ' num2str(states) ];

  % set labels
   xlabel( [ rntext 'Income (x)' ] );
   ylabel( [ 'Probability ' rntext  'Income Exceeds x' ] );
   ttlstart = [rntext 'Incomes ' statestext ': Year '];

  % create matrix with 1 for each personal state to be included
   cells = zeros( size(client.pStatesM) );
   for s = 1:length(states)
     cells = cells + ( client.pStatesM == states(s) );
   end;

  % convert client incomes  to nominal values if required
   if findstr( 'n', plottype ) > 0
      client.incomesM = market.cumCsM .* client.incomesM;
   end;
```

```
% create vector with number of scenarios for each year
  nscens = sum( cells );
% find number of years to plot
  nyrs = sum( nscens > 0 );
% find maximum income
  incomes = client.incomesM .* cells;
  maxIncome = max( max(incomes) );

% set axes for figure
  prop = .01*analysis.plotIncomeDistributionsPctMaxIncome;
  maxIncome = prop * maxIncome;
  propShown  =  analysis.plotIncomeDistributionsProportionShown;
  if propShown < 1.0
     ii = find(cells == 1);
    v =  sort(incomes(ii),'ascend');
    i = fix(propShown * length(v));
    i = max(1,i);
    maxIncome = v(i);
 end;
 ax = [ 0  maxIncome  0  1 ];
  axis( ax );
  hold on;

% set delay change parameter
  delays = analysis.animationDelays;
  delayChange = ( delays(2) – delays(1) ) / (nyrs -1);

% set initial delay
  delay = delays(1);

% set parameters
  % set full color based on states
    clrmat = [ ];
    for s = 1:length(states)
       clrmat = [ clrmat; cmap( states(s)+1, : ) ];
    end;
    clrFull = mean( clrmat, 1 );

  % set shade color
    shade = analysis.animationShadowShade;
    clrShade = shade * clrFull + (1-shade)*[ 1  1  1];
  % set initial delay
    delay = delays( 1 );
  % plot each year's distribution
   for yr = 1 :nyrs

     % find values for y axis
        rows = find( cells( :, yr ) == 1 );
        incomes = client.incomesM( rows, yr );
        yx = 1:1:length(incomes);
        if findstr( 'c', lower(plottype ) ) > 0
            yx = yx / length(yx);
          else
            yx = yx / size(client.incomesM, 1);
        end;

     % compute probability of states and round to one decimal place
        if findstr( 'c', lower(plottype) )>0
           probPstates = length(incomes) / size(client.incomesM,1);
          else
           probPstates = length(incomes) / size(client.incomesM,1);
        end;
        probPstates = round(1000*probPstates) / 10;

     % plot if probability large enough
       if probPstates >=  analysis.plotIncomeDistributionsMinPctScenarios
          plot(sort(incomes,'ascend'), sort(yx,'descend'), 'color', clrFull, 'Linewidth', 3);
          ttl1 = [ ttlstart num2str(yr) ];
          ttl2 = [ num2str(probPstates) ' Percent of Scenarios' ];
         title( {ttl1, ttl2} 'color', 'b');
          pause(delay);
          plot( sort(incomes,'ascend'), sort(yx,'descend'), 'color', clrShade, 'Linewidth',3 );
          delay = delay + delayChange;
       end;

   end; % for yr = 1, nyrs

end
```

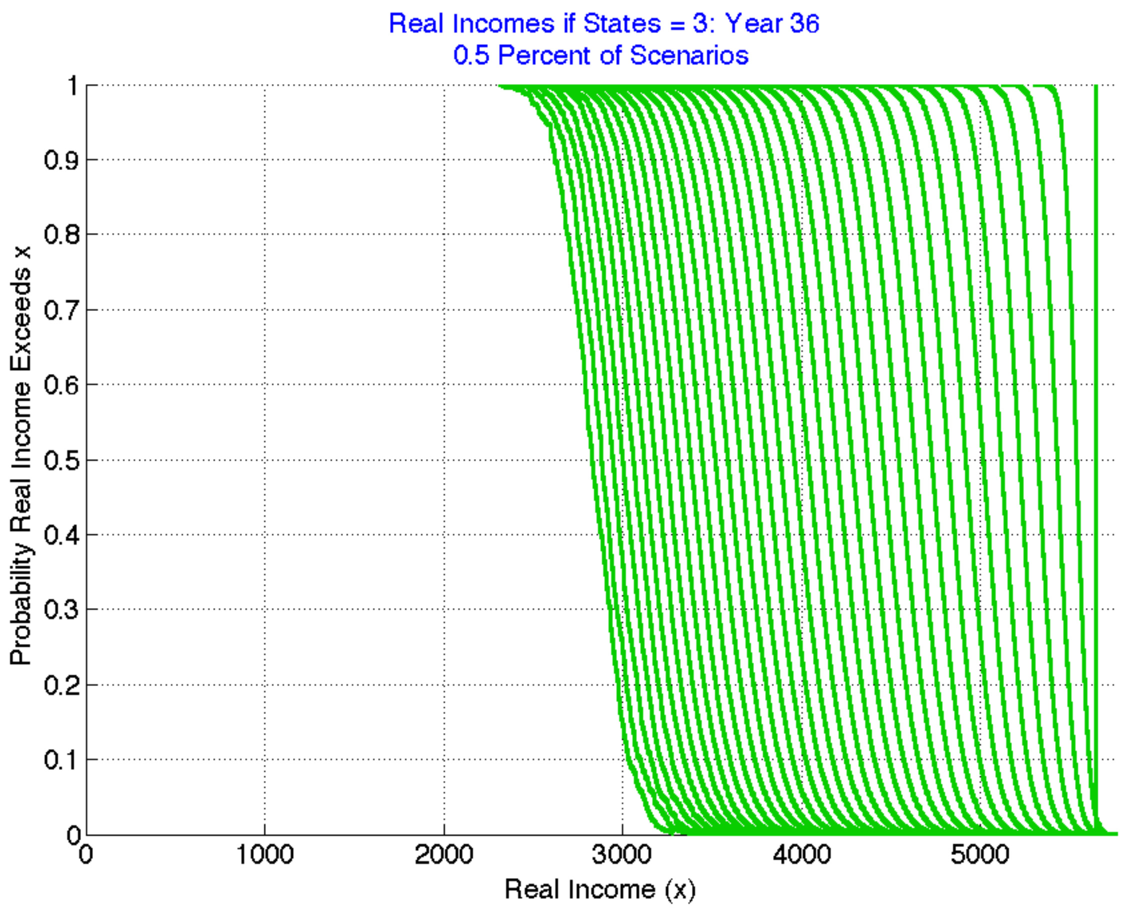
The initial statements set up the figure, initialize a matrix with the set of colors we use in all analyses for the personal states, and sets up labels for the graph. The next statements create a matrix with “1” in each cell that corresponds to the desired personal state or states, then another matrix with the desired type of income in each such cell.

Next, a color is chosen based on the state or states being plotted. If only one personal state has been chosen, its color is used. If more than one state is being shown, a color is determined by averaging the colors of the states in question.

The remaining statements produce the plots, one for each year desired. Of necessity, some housekeeping is required to accommodate the different types of graphs that can be produced, but the procedures are relatively straightforward.

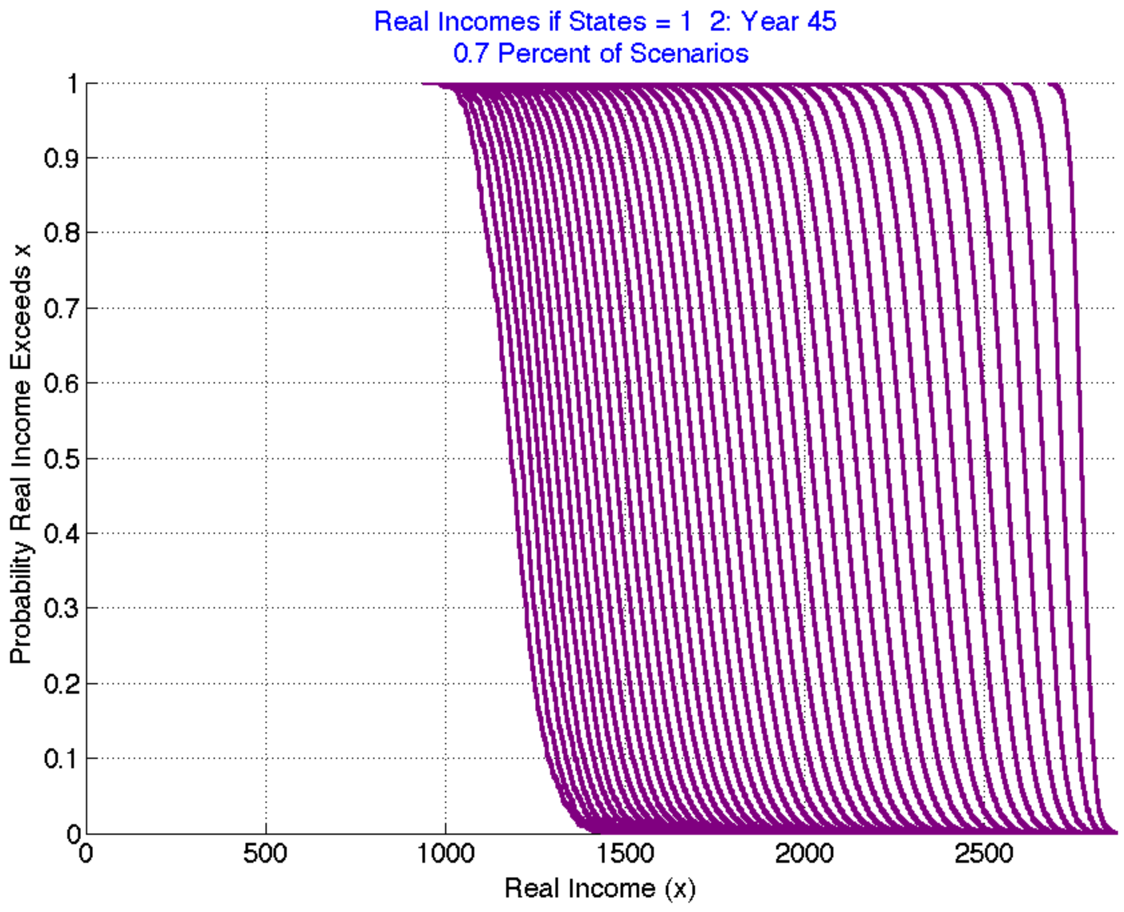
These preliminaries completed, let's turn to examples. We will focus on the prior example of a fixed nominal annuity paying the same amount each year when both Bob and Sue are alive, then half that amount when only one of them is alive. We will not show nominal income graphs since they would be boring: each year's income distribution would plot as a vertical line and all the lines would be at the same location on the x-axis. The following pages show real income distributions for state 3 (both alive) and states 1 and 2 (one alive).

We begin with the conditional graph of real incomes for states in which both are alive. As with the previous animations, we show the results after all the years have been plotted, with equal shading (*analysis.animationShadowShade = 1*) for each distribution.



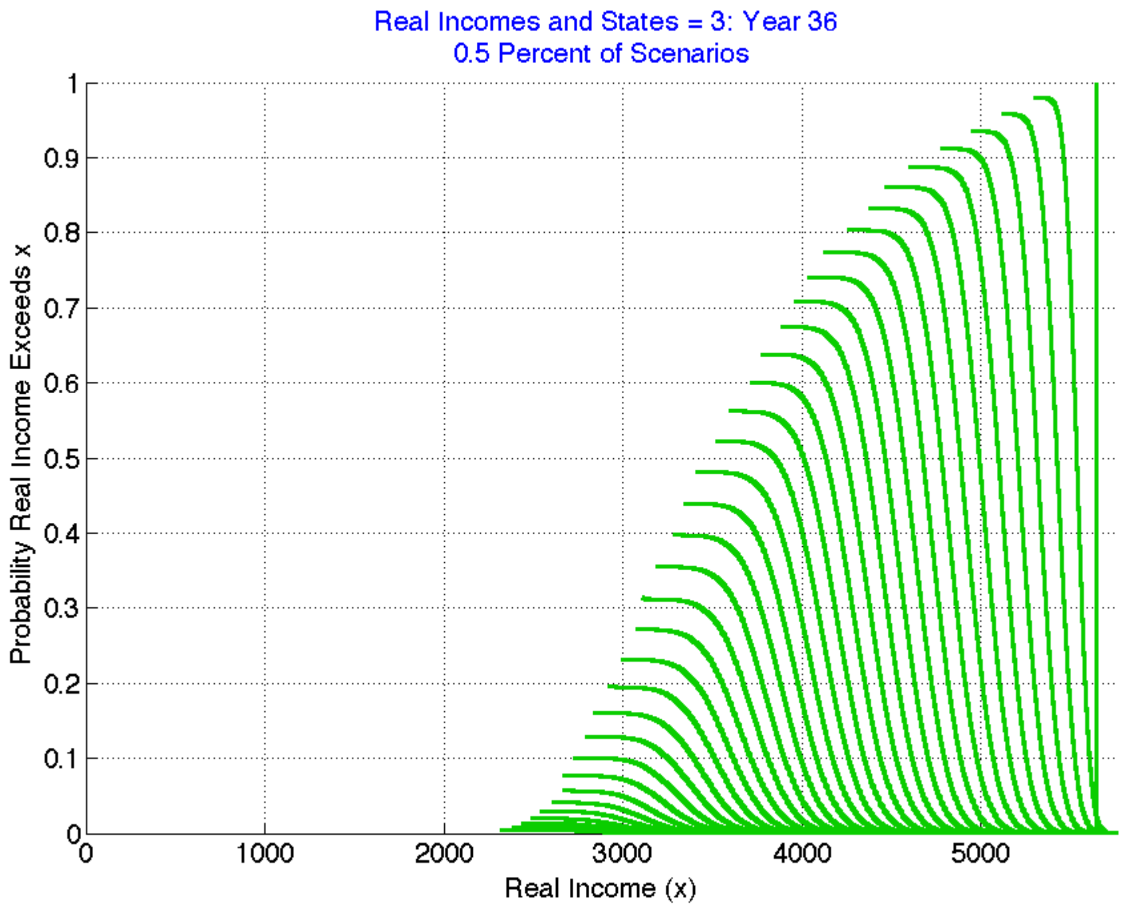
Each year plots as a separate curve, with the first year on the right and each subsequent year to the left of the preceding one. With the exception of the first year's income, which is certain, each plot shows a range of values due to the uncertainty in real income associated with inflation, with the slope dependent on the assumed standard deviation for inflation. Moreover, each year's distribution plots to the left of the prior year, with the distances dependent on the assumed expected rate of inflation. The last year shown is year 36, for which the probability that both Bob and Sue will be alive is just 0.5%. Note that if they make it that far, their annuity could purchase less than half as much in goods and services as it did at inception. It is entirely possible that after examining this graph, Bob and Sue might have chosen to consider a nominal annuity with increasing payments over time. Or, better yet, a real annuity.

Next is the graph for cases in which only one of our protagonists is alive – Bob (state 1) or Sue (state 2). Note first that the plots are all purple, obtained by averaging their two colors – blue for Bob and red for Sue. Note second, that the amounts of income are generally half those that would be obtained if both were alive, due to the terms of the joint and survivor annuity. And, not surprisingly, there are significant probabilities of income being received in these states for periods up to year 45 since it is possible that one of the two could live that long.

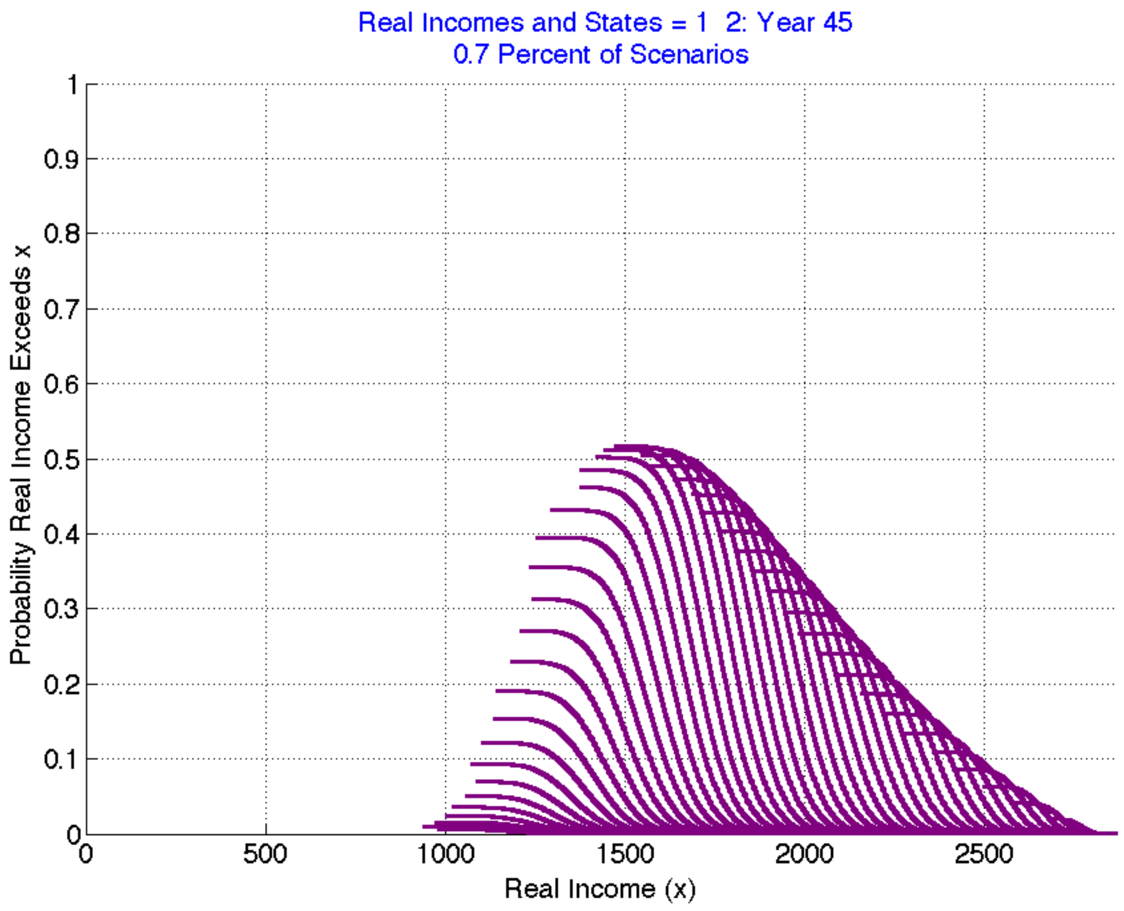


While conditional graphs such as those we have seen can be helpful without watching them develop year by year, some information is lost (for example, the changes in the title indicating the year and probability of income in that year). For this and other reasons, unconditional views may be more informative.

This figure below shows such a graph for income in state 3, when both Bob and Sue are alive. Not surprisingly, each year's plot lies to the left and below that of the prior year. The top of each curve, indicating the probability that Bob and Sue will be alive in the year in question is lower in each subsequent year, ending at 0.005 (0.5 percent) in year 36. The overall result is useful and, some might say, decorative.



A final graph of this type portrays the unconditional results for states 1 and 2. The result, which looks a bit like diminishing waves coming from the sea (to the right) farther and farther inland, is shown below.



The curves reflect the fact that in early years, the probabilities that only one person will be alive are low. The chances then increase and eventually begin to decrease. Meanwhile, the ranges of income fall. In every year, there is uncertainty about real income due to the unknown amount of cumulative inflation up to that point. Perhaps not the most desirable set of prospects, but a fascinating pattern, nonetheless.

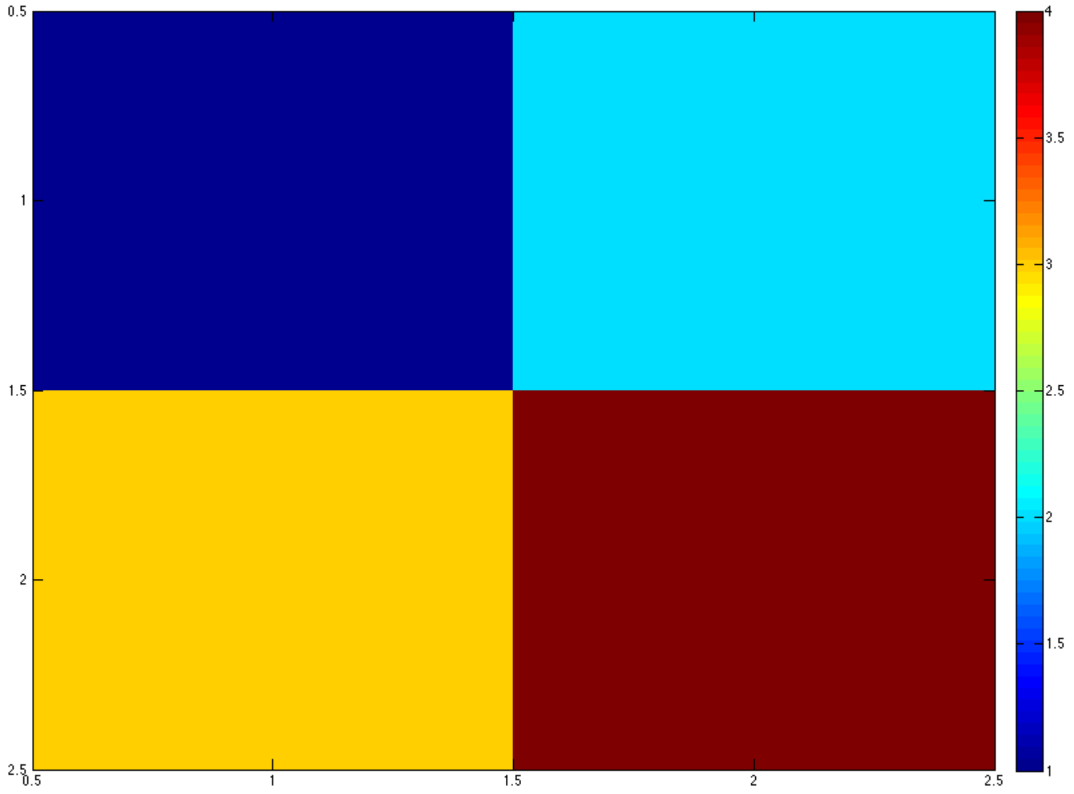
Income Distribution Maps

Animated income distribution graphs have their merits. When one can see them “live”, it is possible to get a sense of the changes in ranges of outcomes as later and later years are shown. But after the process is complete, many of these annual differences may be lost, except in cases (such as the one we have shown) in which there is a separation in the annual distributions and a neat progression as later and later years are plotted. In a standard document or on a static web site, only a “dead” (completed) version can be provided. We now consider an alternative portrayal that uses Matlab's very useful ability to portray the values in a matrix as a *heat map* – a graph in which matrix values are displayed as colors.

The programming statements are simple indeed. If *M* is a two-dimensional matrix, one simply writes:

```
imagsc( M );
colorbar;
```

For example, if M = [1 2; 3 4] the result would be:



The *imagesc* function creates an *image* that is scaled (*s*) and in color (*c*). The *colorbar* function shows the colors being used for the different values in the range of outcomes. These are taken from the current *colormap*, which can be changed if desired.

As usual, we start by adding elements to the analysis data structure in the *analysis_create* function:

```
% plot income maps
   analysis.plotIncomeMaps = 'n';
% plot income maps: set of cases with real or nominal (r/n) and
%   conditional or unconditional (c/u) types
   analysis.plotIncomeMapsTypes = { 'ru'  'rc' };
% plot income maps: sets of states (one set per graph)
   analysis.plotIncomeMapsStates = { [3]  [1 2] };
% plot income distributions: minimum percent of scenarios
   analysis.plotIncomeMapsMinPctScenarios = 0.5;
% plot income maps: percent of maximum income plotted
   analysis.plotIncomeMapsPctMaxIncome = 100;
```

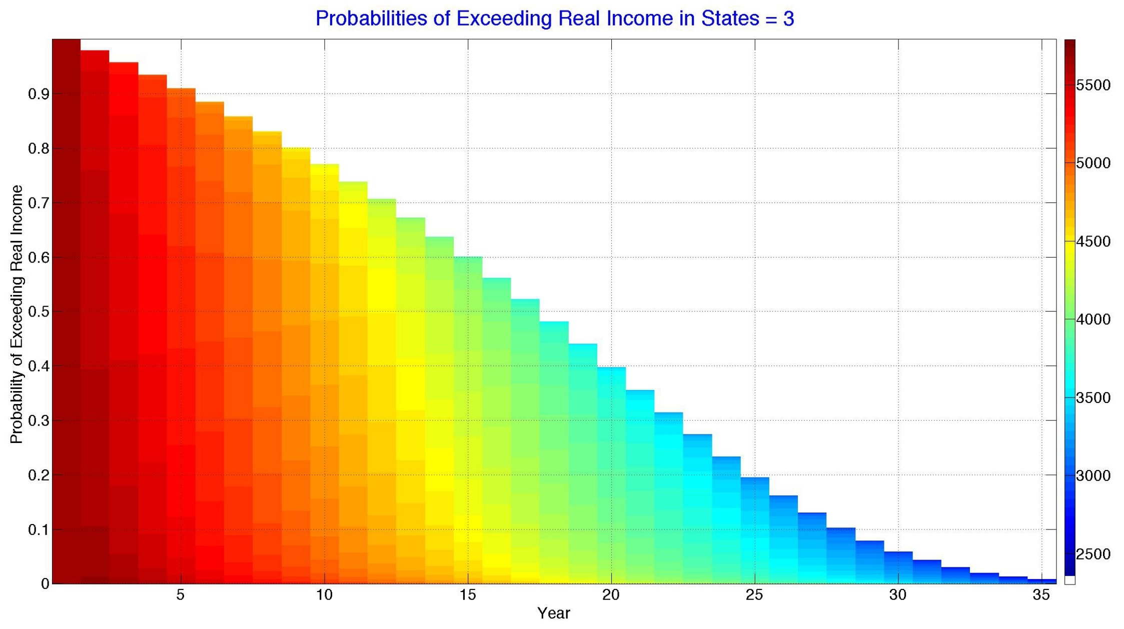
The pattern is somewhat similar to that used for income distributions. Graph types can use real (*r*) values or nominal (*n*) values, and conditional *(c)* or unconditional *(u)* results. As with the income distributions, different sets of personal states can be included. The plots can be limited to years in which there are a minimum percent of relevant scenarios. Finally, the maximum percentage of the income to be plotted can be specified; in this case all greater values will be plotted as if they equaled the resulting maximum amount.

The next step is to add statements to the *analysis_process* function that will call an external function that we will name *analPlotIncomeMaps.* Not surprisingly, the statements are very similar to those used for the animated plotting of the income distributions. Here they are:

```
% analysis: plot income maps
 if analysis.plotIncomeMaps == 'y'
    % find states;
      states = analysis.plotIncomeMapsStates;
    % find types
      types = analysis.plotIncomeMapsTypes;
    % create figures
      for i = 1 : length( types )
         for j = 1 : length( states )
            % create Figure
               analysis = createFigure( analysis, client );
            % call external function analPlotSurvivalRates
               analPlotIncomeMaps( analysis, client, market, types{i}, states{j} );
             % process figure
               analysis = processFigure( analysis );
         end; %j
      end; %i
 end;
```

To produce a map there remains only the task of creating the *analPlotIncomeMaps* function. As before, we will include it in its entirety, then summarize the tasks it performs, highlighting any new programming constructs. But first, an example.

Here is the map for ranges of unconditional annual real incomes for our constant nominal joint and survivor fixed annuity for the states in which both Bob and Sue are alive.

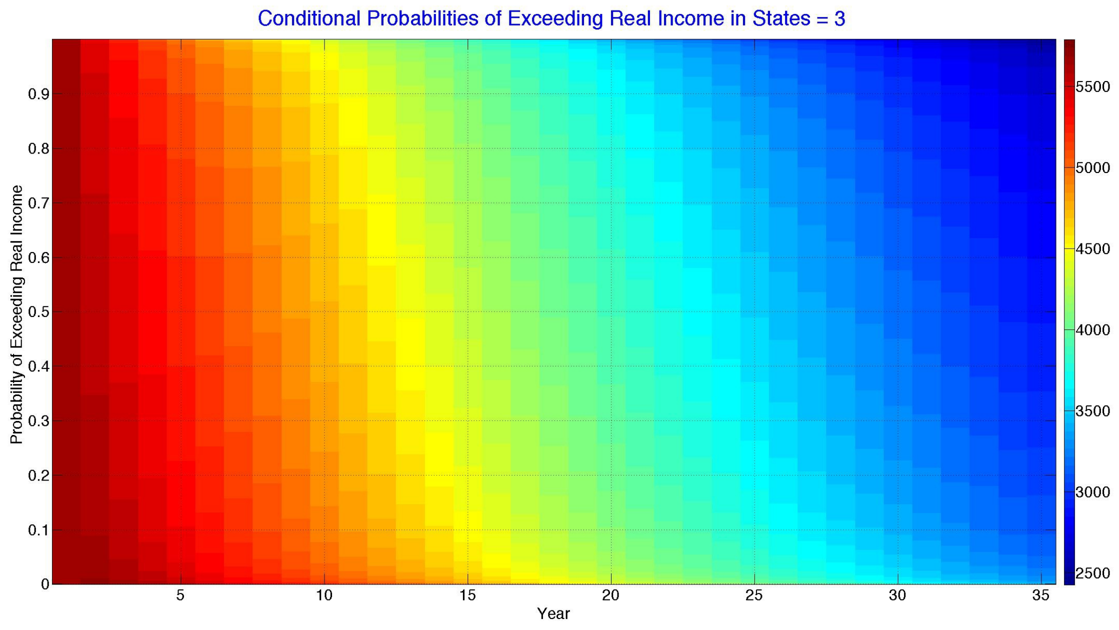


One way to think about this graph is to consider each column (year) as a version of the corresponding distribution plot for that year (as produced earlier) rotated 90 degrees – holding the vertical axis in place and pulling the right end of the horizontal axis up and to the left until the horizontal axis points toward the viewer. We then use an appropriate color to indicate the height of each point. Then we repeat the process for each year, putting each subsequent graph to the right of that for the prior year. This description may or may be helpful, and some viewers will be perfectly happy to take this graph on its own (and maybe even skip the animated distribution graph entirely). Given the limitations of two-dimensional media, we will do so at places in future chapters.

There is a lot of information in this graph. First, the heights of the bars show the probabilities that the state or states in question will occur (in this sense, they contain some of the information in the survival probabilities graph). Next, the colors in each bar show the likelihoods of exceeding the corresponding real incomes shown in the *colorbar*. Looking vertically for a given year one can thus see the range of possible incomes.

Looking across horizontally shows the year-by-year changes in the distributions. And following a given color across the graph from left to right shows the changing probabilities of being alive and exceeding the income associated with that color.

The conditional version of the income map is helpful if one wants to compare the ranges of income without taking into account the probabilities of the occurrence of the included state or states. Here is the graph for the annuity being analyzed:



The effects on real incomes from accepting a constant nominal income are very clear in this version.

Now to details. For those who are interested, here is the function:

```
function analPlotIncomeMaps( analysis, client, market, plottype, states )
  % plots income images using personal states in vector states

  % initialize graph
    set( gcf, 'name', ['Income Images ' plottype] );
    set( gcf, 'Position', analysis.figPosition );
    grid on;
  % make plottype lower case
    plottype = lower( plottype );

  % set real or nominal text
    if findstr( 'n', plottype ) > 0
        rntext2 = 'Nominal ';
      else
        rntext2 = 'Real ';
    end;
    if findstr( 'c', plottype ) > 0
        rntext1 = 'Conditional';
      else
        rntext1 = '';
    end;

  % set states text
    statestext = [ 'States = ' num2str(states) ];

  % convert client incomes  to nominal values if required
    if findstr( 'n', plottyp e) > 0
       client.incomesM = market.cumCsM .* client.incomesM;
    end;

  % create matrix with 1 for each personal state to be included
    nscenarios = size( client.pStatesM, 1 );
    cells = zeros( size(client.pStatesM) );
    for s = 1:length(states)
      cells = cells + ( client.pStatesM == states(s) );
    end;

  % make matrix with incomes for included personal states
   incomes = cells .* client.incomesM;

  % find cells with included personal states
    ii = find( cells > 0 );
```

```
% find minimum and maximum incomes for included personal states
   mininc = min( incomes(ii) );
   maxinc = max( incomes(ii) );

% find last year with sufficient included states
  [nscen,nyrs] = size( incomes );
  numstates = sum( cells > 0 );
  minprop = analysis.plotIncomeMapsMinPctScenarios;
  minnum = ( minprop / 100 ) * nscen;
  lastyear = max( (numstates > minnum) .* ( 1:1:nyrs ) );

% reduce matrices to cover only included years
  incomes = incomes( :, 1:lastyear );
  cells =  cells( :, 1:lastyear );

% create colormap
  colormap( 'default' );
  map = colormap;
  map( 1, : ) = [ 1 1 1 ];
  colormap( map );
% put a lower value in each excluded personal state
  ii = find( cells < 1 );
  incomes(ii) = mininc - 1;

% make changes if map is to be conditional
 if findstr( 'c', plottype ) > 0  % convert to conditional incomes
   incs = incomes( :, 1:lastyear );
   condincs = [  ];
   for yr = 1:size(incs,2)
       % extract values for chosen personal states
         yrincs = incs( :, yr );
         yrcells = cells( :, yr );
         ii = find( yrcells > 0 );
         vals = yrincs( ii );
       % create full vector of values greater than the minimum
         num = length(vals);
         m = vals * ones( 1, ceil(nscen/num) );
       % extract the first nscen values as a vector
         v =  m( 1: nscen );
         if size(v,2) > 1; v = v'; end;
       % add to conditional incomes matrix
         condincs = [ condincs v ];
   end; % for yr = 1:size(m,2)
   incomes = condincs;
   colormap( 'default' );
 end; % if findstr( 'c', plottpe ) > 0
```

```
  % truncate incomes above percentage of maximum income
    prop = .01 * analysis.plotIncomeMapsPctMaxIncome;
    maxinc = prop * max(max( incomes(:,1:lastyear) ) );
    incomes( :, 1:lastyear ) = min( maxinc, incomes(:,1:lastyear) );

  % plot
   imagesc(sort( incomes(:, 1:lastyear), 'ascend') );
   grid;
   cb = colorbar;
   set( cb, 'Fontsize', 30 );
   set( gca, 'Fontsize', 30 );
   set( gca, 'YtickLabel', [.9 .8 .7 .6 .5 .4 .3 .2 .1 0] );

  % set labels
   xlabel( [ 'Year' ], 'Fontsize', 30 );
   ylabel( [ 'Probability of Exceeding ' rntext2  'Income ' ], 'Fontsize', 30 );
   ttl = [ rntext1 ' Probabilities of Exceeding ' rntext2 'Income in ' statestext ];
   title( ttl, 'Fontsize', 40, 'color', 'b' );

end
```

The first portions of the function are similar to those for the income distributions graph: they initialize the graph's name and position and set some of the text to be used for labels, depending on the condition, type of income (real or nominal) and states to be included. The next block converts real values to nominal, if required.

The next set of commands creates a matrix with the incomes for the personal states to be included, finds the minimum and maximum incomes, the last year with a sufficient number of incomes, then revises the income and personal state matrices to include only the desired number of years.

The next portion is designed primarily for plotting an unconditional version of the results; some of its actions will be reversed if a conditional version is to be shown. In any event, this section selects a colormap equal to the default global matrix *colormap,* which has 64 different colors, then changes the first one to white ([1 1 1]), storing the result in the global matrix. It then puts a value slightly lower than the lowest included income in every cell that is to be excluded so each such cell will be shown in white if needed.

An aside is in order here. The use of a lower value to obtain white space is, at the least inelegant and could even be termed a *kludge*, defined by wikipedia as:

> *.. a workaround or quick-and-dirty solution that is clumsy, inelegant, inefficient, difficult to extend and hard to maintain. It is a rough synonym to the term "jury rig". This term is used in diverse fields such as computer science, aerospace engineering, internet slang, evolutionary neuroscience and government.*

On the other hand, one person's kludge may be another's clever programming method. Reader's choice.

Now, to return to the description of this function.

The next section is used if the map is to be conditional. The key problem to be solved is that the number of scenarios with incomes differ from year to year, depending on personal states. For example, if only incomes for personal state 3 are to be shown, all the scenarios will be relevant for year 1, but only a subset for, say, year 20. But the graphics function *imagesc( )* requires a matrix with values in every cell. Our approach deals with this on a year-by-year (column-by-column) basis. For each year, the relevant incomes for the chosen personal states are extracted, giving a vector with potentially fewer values than there are scenarios. This vector is then used to create a new vector with at least as many values as there are scenarios, in effect repeating the values as many times as needed to make a new vector as long or longer than the number of scenarios. Then the first *nscen* values are used to create a new vector. All such vectors are used to produce an income matrix with the same dimensions as the original one, which can then be plotted.

This may well qualify as a kludge, but it is effective. The only possible concern should be the likely use of only a subset of relevant incomes for the last part of the vector. Fortunately the procedure should be unbiased, since the incomes have been generated using randomly chosen market returns, etc.. Although inelegant, the approach provides useful portrayals of the ranges of possible incomes.

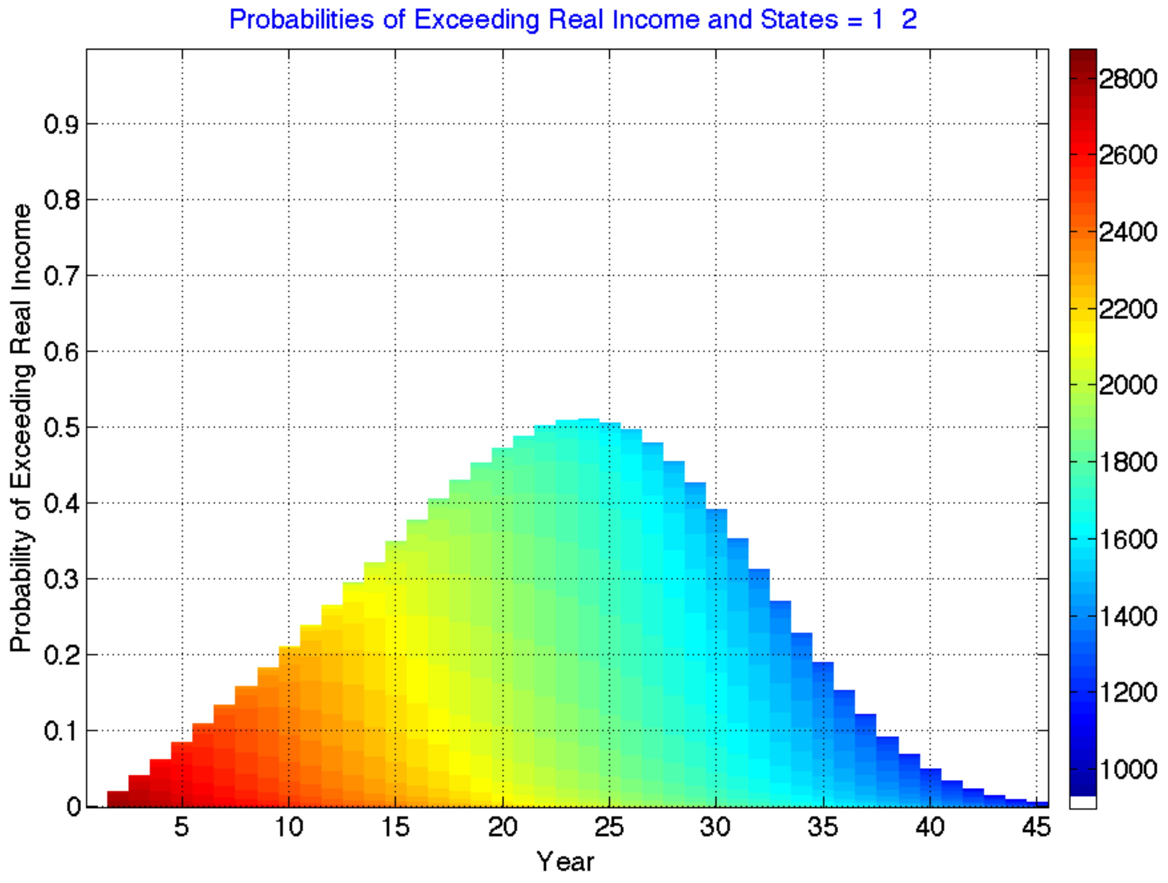
One last task for a conditional map: since no white space is needed, we use the default colormap.

The global *colormap*, which may be changed by this function, will be used for both the map and the associated *colorbar*. And any subsequent changes will change whatever figure is the *current figure;* for this reason it is important when creating a subsequent figure that the *colormap* be set to its default value. As shown in Chapter 11, the *analysis_process* function does this whenever a new figure is created (and the mystery described there is now resolved).

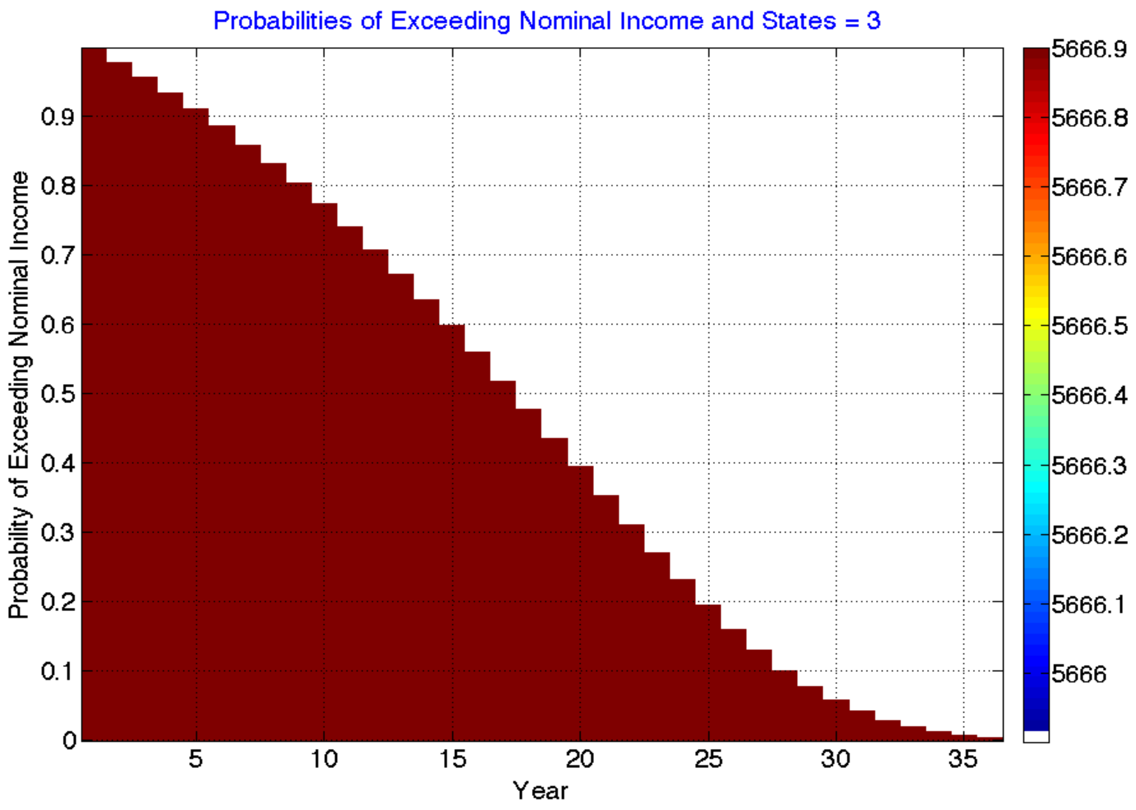
The remaining statements replace incomes above the specified proportion of the maximum with the maximum, then create the image using the *imagesc* function with a matrix of sorted incomes for each year, with appropriate labels. And the job is done.

The tedious description of the *analPlotIncomeMaps* function complete, we return to matters of substance, considering three other possible income distribution maps.

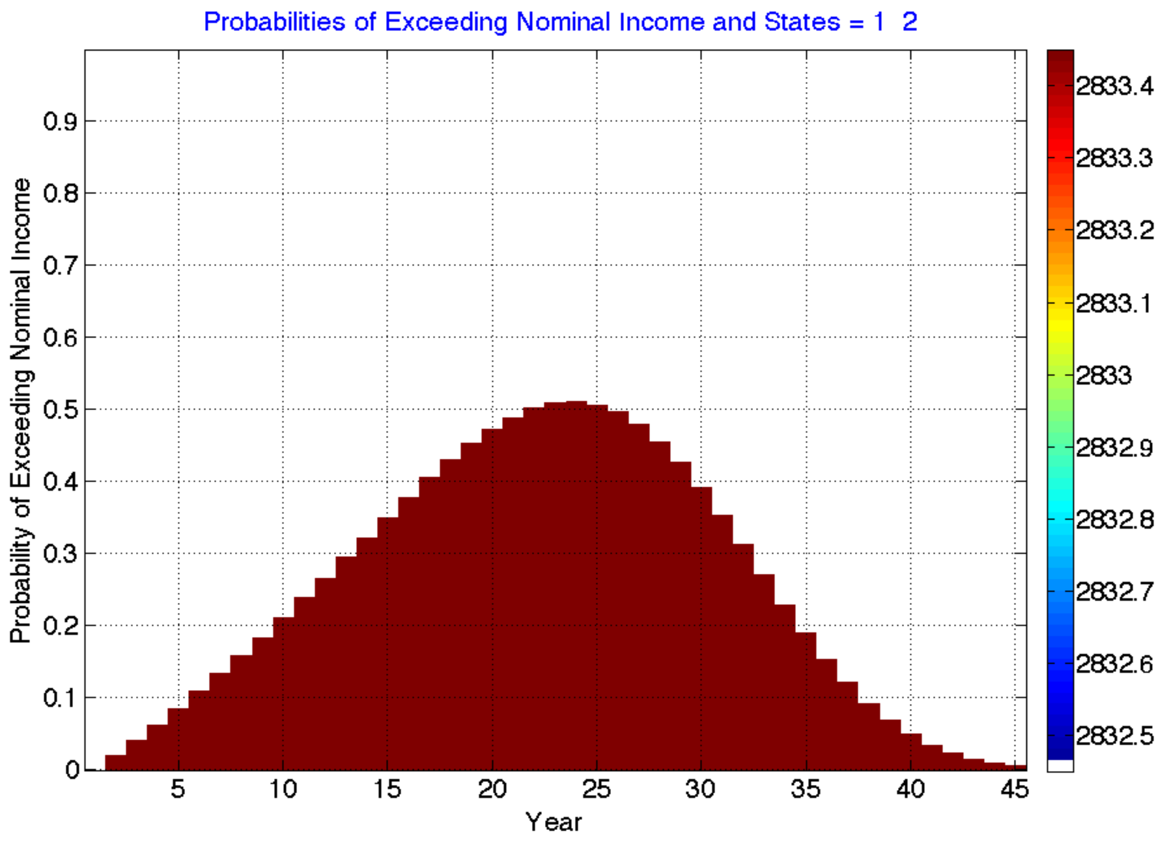
The figure below shows the unconditional real incomes for personal states 1 and 2. As we have seen before, the probability that only one person will be alive is small in the early years, increases in the middle years, then falls in the later years. Moreover, with our annuity's constant nominal income, real income tends to be smaller, the later the year (as shown by the movement from red to blue as one looks from left to right). As before, there is uncertainty about the income in any given year (illustrated by the variation in color as one looks up and down in a given bar). But this uncertainty is relatively small, as can be seen by checking the actual incomes for different colors, shown in the *colorbar* to the right of the diagram.



Nominal values for our annuity are, of course all the same for personal state 3 (both Bob and Sue alive) and half as much for state 1 (Bob alive) or 2 (Sue alive). The graphs are, accordingly far less colorful. Here is the one for personal state 3:



And the one for personal states 1 or 2.

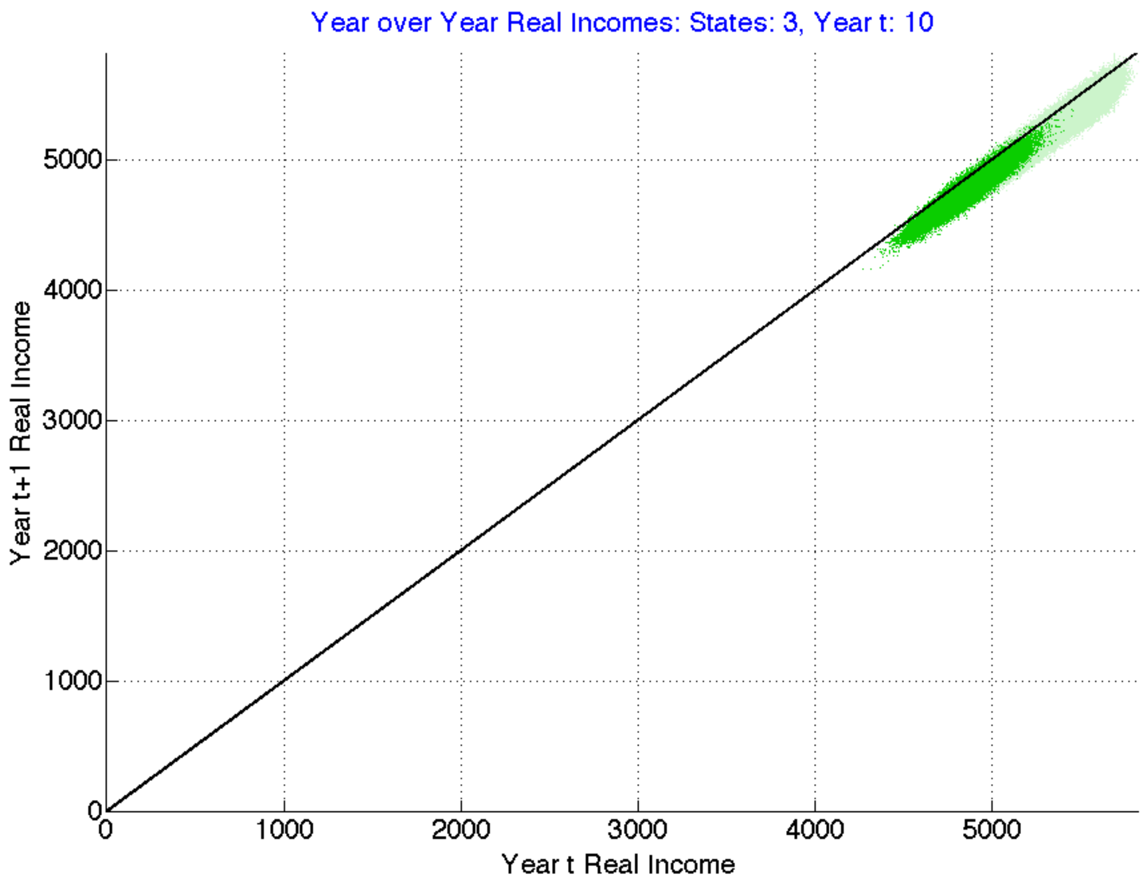


Perhaps surprisingly, the time required to produce all four of these graphs was modest – less than 9 seconds on the author's Macbook Pro.

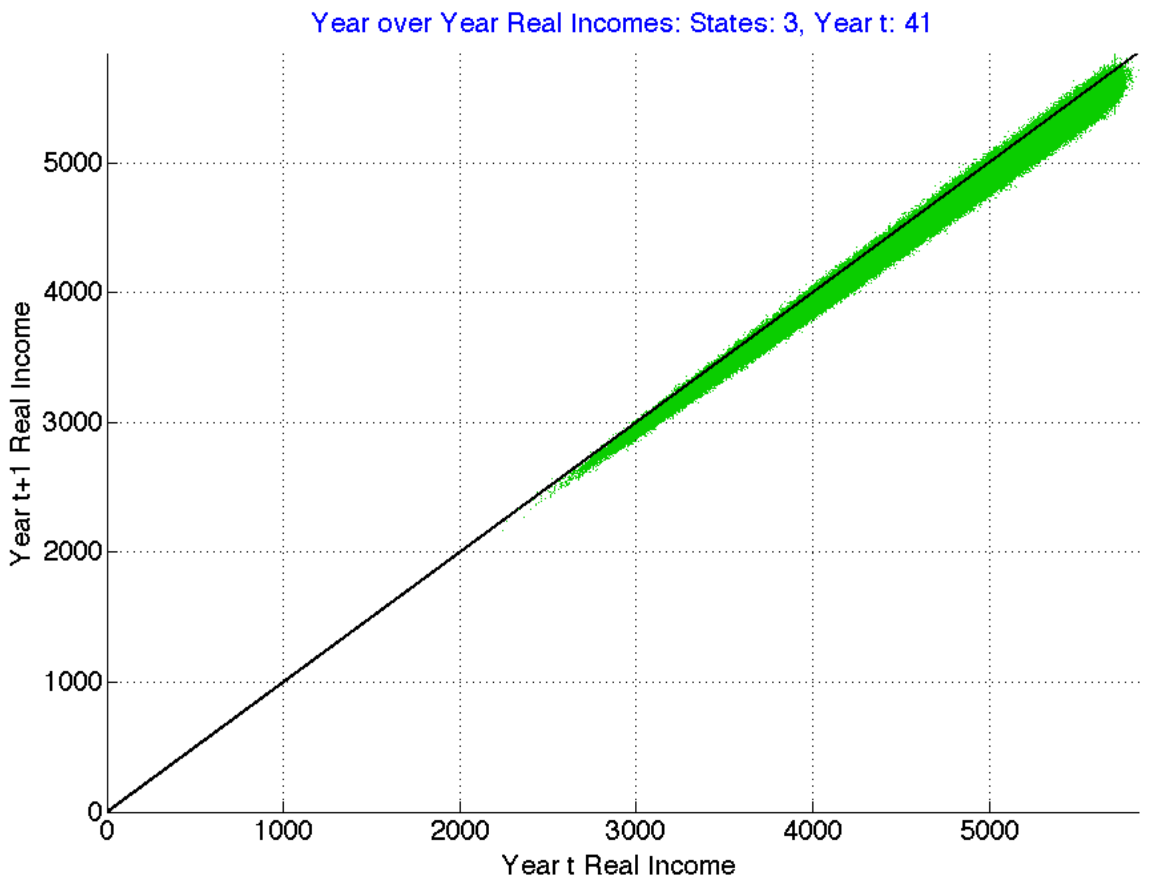
Year-over-Year Incomes

The animated income distribution graphs and the income distribution maps contain a great deal of information about possible ranges of income in future years for a given strategy or combination of strategies. For many retirees, this should suffice for evaluating a strategy and for comparing it with alternative approaches. To return to the formalities of Chapter 9, such people can be considered to have *time-separable utility functions*. But, as we have indicated, some retirees also care about the sequence of incomes from year to year. Our scenario plots provide information that can help address this concern, but it is impractical to expect a retiree to watch thousands of sequences of possible future income for one strategy, then watch thousands of sequences for another strategy, then make a choice between the two alternatives, let alone repeat the process with many other strategies. Instead we offer as a partial solution, *year over year income graphs*.

The figure below provides an example. It shows the results for our fixed nominal annuity after 10 yearly incomes have been plotted. The darker points show real incomes in year 10 (on the horizontal axis) and year 11 (on the vertical axis) for all scenarios in which both Bob and Sue are alive in each of the two years (personal state 3). Not surprisingly, there is considerable variation in real incomes in each year due to variations in cumulative inflation over the prior years. Moreover, as shown by the position of the points relative to the 45-degree line, in most scenarios the income in year 11 is less than that in year 10, due to the effects of inflation in the intervening year, which we have assumed to have an expected value of 2%. Moreover, the variation from year to year is relatively small, since our default assumption is that the standard deviation of annual inflation is only 1% .



The next figure shows the results when all the years with sufficient scenarios have been plotted (using a shade parameter of 1.0). The points for later years tend to lie below and to the left of those for earlier years, due to the ravages of inflation on the purchasing power of our fixed annuity, but there is some overlap from year to year. At first glance, the diminution of the vertical spread of the points as one moves to the left may seem mysterious, but the explanation is relatively simple. In this case the difference between the real income in one year and that in the next is determined by the rate of inflation in the intervening year. And this has a multiplicative effect on real income. Thus, for a given level of inflation, the lower the beginning real income, the smaller is the absolute change in real income in the subsequent year.



Now the programming details, for those committed to getting down in the Matlab weeds.

First, we add to the *analysis_create* function four elements to serve as parameters:

```
% plot year over year incomes
   analysis.plotYOYIncomes = 'n';
% plot year over year incomes -- real or nominal (r/n)
   analysis.plotYOYIncomesTypes = { 'r'  'n' };
% plot year over year incomes -- sets of states (one set per graph)
   analysis.plotYOYIncomesStates = { [ 3 ]  [ 1  2 ] };
% plot year over year incomes -- include zero (y/n)
   analysis.plotYOYIncomesWithZero = 'y';
```

As with previous income plots, we allow for real or nominal values and for different personal states or sets of such states. The final parameter dictates whether the graph's origin should include zero income in each year. The default setting is yes (*'y'*) in order to provide visual perspective for the relative magnitudes of the plotted incomes. However, for diagnostic analyses, it may be useful to change the value to *'n'* to enlarge the data area for closer inspection.

The statements to be added to the *analysis_process* function follow the same approach used in the prior case:

```
% analysis: plot year over year incomes
   if analysis.plotYOYIncomes == 'y'
      % find states;
        states = analysis.plotYOYIncomesStates;
      % find types
        types = analysis.plotYOYIncomesTypes;
      % create figures
        for i = 1:length( types )
          for j = 1:length( states )
            % create Figure
              analysis = createFigure( analysis, create );
            % call external function analPlotSurvivalRates
              analPlotYOYIncomes( analysis, client, market, types{i}, states{j} );
            % process figure
              analysis = processFigure( analysis );
          end; %j
        end; %i
   end;
```

No surprises here.

Following our conventions, the external function that does the hard work is called *analPlotYOYIncomes.* Here it is:

```
function analPlotYOYIncomes( analysis, client, market, plottype, states )
  % plots income images using personal states in vector states

  % initialize graph
     set (gcf, 'name', ['YOYIncomes ' plottype] );
     set( gcf, 'Position', analysis.figPosition );
     grid on;
  % make plottype lower case
     plottype = lower( plottype );

  % set real or nominal text
     if findstr( 'n', plottype )  >  0
          rntext = 'Nominal ';
       else
          rntext = 'Real ';
     end;

  % set states text
     statestext = [ 'States: ' num2str(states) ];

  % convert client incomes  to nominal values if required
     if findstr( 'n', plottype ) > 0
         client.incomesM = market.cumCsM .* client.incomesM;
     end;

  % set labels
    xlabel( [ 'Year t '  rntext  'Income ' ] );
    ylabel( [ 'Year t+1 ' rntext  'Income ' ] );
    ttl = [ 'Year over Year ' rntext  'Incomes: '  statestext ', Year t: ' ];

  % create matrix with 1 for each personal state to be included
     cells = zeros( size(client.pStatesM) );
     for s = 1:length( states )
        cells = cells + ( client.pStatesM  ==  states(s) );
     end;

  % find last year with income for personal states
     nyrs =  max( find(sum(cells)  >  0) );
  % modify matrices
     incs = client.incomesM( :,  1:nyrs );
     cells = cells( :, 1:nyrs );
```

```
% set axes
 ii = find( cells  >  0 );
 maxval = max( incs(ii) );
 minval = min( incs(ii) );
 if analysis.plotYOYIncomesWithZero == 'y'
    minval = 0;
 end;
 axis( [ minval maxval minval maxval ] );

% initialize plot
 grid on;
 hold on;

% set colors for states 0,1,2,3 and 4
% orange; red; blue; green; orange;
 cmap = [ 1 .5 0 ; 1 0 0; 0 0 1; 0 .8 0; 1 .5 0 ];
% set full color based on states
 clrmat = [ ];
   for s = 1:length( states )
      clrmat = [ clrmat; cmap( states(s)+1, : ) ];
   end;
 clrFull = mean( clrmat, 1 );
% set shade color
 shade = analysis.animationShadowShade;
 clrShade = shade * clrFull + (1-shade)*[ 1 1 1 ];

% set delay change parameter
 delays = analysis.animationDelays;
 delayChange = ( delays(2) – delays(1) ) / ( nyrs -1 );
% set initial delay
 delay = delays(1);

% plot 45 degree line
  plot( [minval maxval], [minval maxval], 'Linewidth', 1, 'color', 'k' );
  % plot incomes
    for col = 2 : nyrs
      ttl1 = [ ttl num2str(col)  '  ' ];
      title( ttl1,  'Fontsize', 20, 'color', 'b' );
      col1 = col - 1;
      col2 = col;
      cellmat = cells( :,  col-1:col );
      ii = find( sum( cellmat, 2 )  >= 2 ) ;
      plot ( incs(ii,col-1), incs(ii,col), '.' , 'color' ,  clrFull, 'Linewidth', 2 );
      plot ( [minval maxval], [minval maxval], 'Linewidth', 2, 'color', 'k' );
      pause ( delay );
      delay = delay + delayChange;
      plot ( incs(ii,col-1), incs(ii,col), '.', 'color' , clrShade, 'Linewidth', 2 );
    end;

end
```

Note that the 45-degree reference line is redrawn after each set of points is plotted to insure that it remains visible throughout. Most of the remaining sections of the function are similar to or the same as those in one or more of the functions described earlier in this chapter. Thus the points in the diagram are given a color determined by the personal state or states included, the delays between pairs of years are determined by the delay parameters in the analysis data structure, and the shade to be used for points prior to the one currently being shown is determined by the associated parameter in the analysis data structure.

The rest of the statements in the function perform tasks similar to those in prior functions and need no additional description here.

Videos

A useful way to provide a set of animated graphs (plus others if desired) is to use a computer's screen capture software to create a video recording while the script is being run. The author's macbook includes *Quicktime,* which makes this possible with little effort. Afterwords, it is straightforward to trim an excess material from the beginning or end of the video. Unfortunately, Apple uses a proprietary format for quicktime videos so it may be desirable to change from a system's chosen format to one that is compatible with a wide variety of operating systems. The videos provided this and subsequent chapters were converted from the quicktime format to an *MP4* format using third-party software.

Here is a link to a video of some of the graphs described in this chapter:

www.stanford.edu/~wfsharpe/RISMAT/SmithCase_Chapter12.mp4

The script that created the material (*SmithCase_Chapter12.m*) is also provided in the directory that contains the RISMAT functions.

Presentation of animated and other graphs in a video format allows users to speed up the display of information, freeze a frame to study a particular display, move back to review a previous image, or skip forward. It also allows for display without any reliance on the availability of MATLAB software.

Values

Our discussion of analytic tools focused on incomes and fees *per se* is now complete. But there is more to be analyzed. The next chapter covers calculation and display of measures of the *value* of incomes and fees, an understanding of which we believe to be essential for fully evaluating alternative ways to finance retirement.