Retirement Income Analysis
with scenario matrices

William F. Sharpe

13. Values

One of the key features of our approach to the analysis of retirement income strategies is the computation of present values. Each of the cells in our *market.pvsM* matrix contains the present value of $1 received in that cell's scenario and year. And each of the cells in our *market.ppcsM* contains the associated present value divided by the probability that the associated scenario and cell will actually occur.

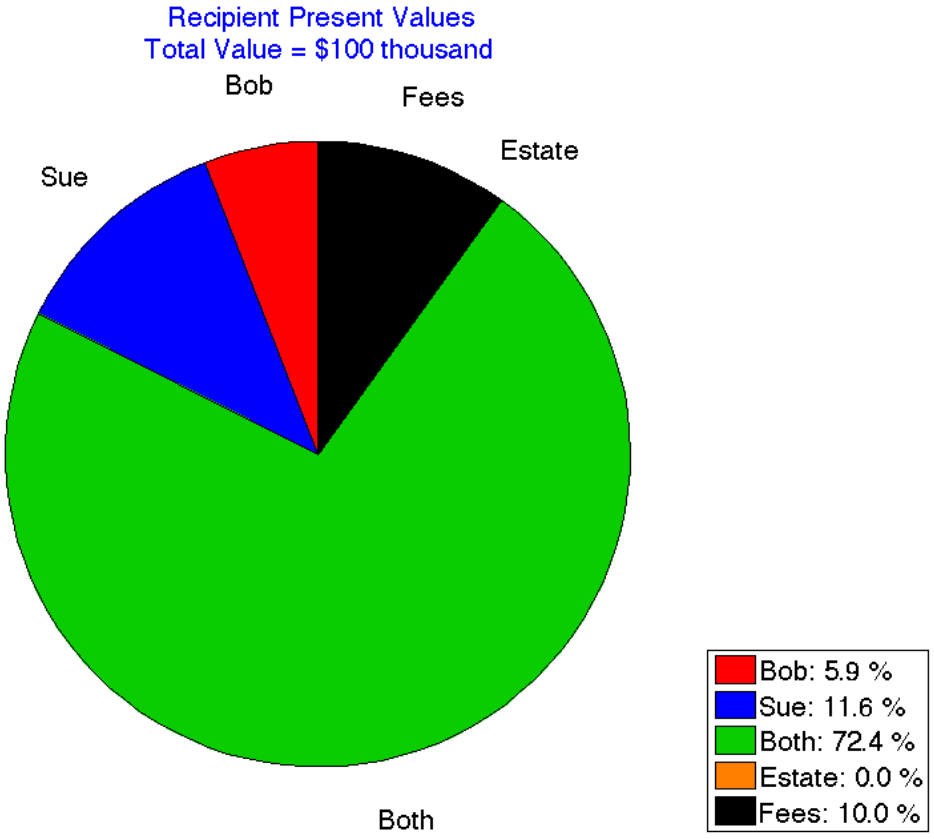
Consider the matrix of present values. We can compute the present value of any set of incomes or fees by simply multiplying each relevant cell by the associated present value and summing all the results. For example, assume the incomes and present value matrices have *nscen* rows (scenarios) and *nyrs* columns (years). Let *cellsM* be a matrix of the same size with a *1* in each relevant cell and a *0* in every other cell. Then the present value of all the relevant incomes will be:

```
PV = sum( sum( cellsM .* market.pvsM .* client.incomesM ) );
```

In effect, for each cell we multiply (a) the indicator (0 or 1) in the first matrix by (b) the present value per dollar in the second matrix and (c) the income in the third matrix; then we add all the results. Voila! The total present value of all the relevant possible incomes.

Recipient Present Values

The figure below shows the result when this procedure was followed for our fixed nominal annuity for each of the personal states (Bob alive, Sue Alive, Both alive, payments to their Estate) and, separately, for the matrix of fees.



In this case there are no payments to the estate – one of the features (or drawbacks, depending on your point of view) of annuity strategies.

Neither the total present value (shown in the title) nor the percentage taken by fees should be a surprise. We created the annuity by investing $100 thousand and assumed a 10% fee at the beginning of year 1. However, in many cases one or both of these present values is not easily computed without the availability of a full valuation model such as the one we have created.

Note that each of these present values takes into account three aspects of any possible income: for every time and state: if an income is paid, the amount that would be paid, and the value of $1 in that time and state. The resulting total values thus take into account mortality, the retirement income strategy as well as time-dependent and state-dependent market valuations.

Of particular interest is the fact that over 70% of the total present value is associated with possible payments made when both Bob and Sue are alive. This, in turn, reflects the substantial likelihood that they will both survive for many years and the fact that income received in earlier years years (in which they both are likely to be alive) tends to be worth more than income received in later years (when only one of them is likely to be alive). And, of course, the total present value of money that might be paid when only Bob is alive is smaller than the corresponding value for Sue, since Bob is both older and male.

This is not an attractive situation for any children that Bob and Sue might have, or for any of their favorite charities, since they have chosen an income strategy that leaves nothing to their estate. As we will see, there are many strategies in which the present value of the estate is significant, even though when the future actually plays out the amount received by the estate may be large, small or even possibly zero.

One of the important aspects of pie charts showing the composition of present value is the percent of discretionary funds taken by fees. Here there is no surprise. But as we will see, many strategies involve continuing expenses that subtract from the recipient's incomes and/or estates. It is important to obtain *ex ante* estimates of the extent to which retirees' wealth is likely to go to financial providers as overhead rather than to the recipients who have accumulated it before retirement. The Recipient Present Value pie chart does this with great efficiency.

To make this happen, we add one statement to the *analysis_create* function:

```
% plot recipient present values -- y (yes) or n (no)
    analysis.plotRecipientPVs  =  'n';
```

And a few to the analysis_process function:

```
% analysis: plot recipient present values
    if analysis.plotRecipientPVs == 'y'
      % create figure
       analysis = createFigure( analysis  );
      % call external function analPlotRecipientPVs
       analPlotRecipientPVs(  analysis, client, market );
      % process figure
       analysis = processFigure( analysis );
    end;
```

The *analPlotRecipientPVs* routine uses many of the approaches in prior external functions plus some new ones required to provide legends and other features. For completeness, here it is:

```
function analPlotRecipientPVs( analysis, client, market );
  % plot recipient present values as pie or bar chart
  % called by analysis_process function
  % compute values for state incomes
    pvs = [ ];
    for state = 0:4
      ii = find( client.pStatesM == state );
      pv = market.pvsM(ii)' * client.incomesM(ii);
      pvs = [ pvs  pv ];
    end;
  % add states 0 to state 4 for estate total
    pvs = [ pvs(2:4) pvs(1)+pvs(5) ];
  % compute fees
    fees = sum( sum( market.pvsM.*client.feesM ) );
  % add fees to present values
    pvs = [ pvs fees ];
  % compute total value and create string in $thousands
    totalVal = sum( pvs );
    totalValStg = (num2str( round( totalVal ) / 1000 ) );
  % if any value is zero change to small positive value
    for i = 1:length(pvs);
      if pvs(i) == 0;  pvs(i) = 0.00001; end;
    end;
  % compute proportions
   props = 100*( pvs / sum(pvs) );

  % create chart
    set(gcf, 'name',  'RecipientPresent Values' );
    set(gcf, 'Position', analysis.figPosition );
  % create legends
    legends = {  };
    legends{1} = [ client.p1Name ': ' num2str( props(1), '%7.1f' ) ' %' ];
    legends{2} = [ client.p2Name ': ' num2str(props(2), '%7.1f' ) ' %'];
    legends{3} = [ 'Both: '   num2str( props(3),'%7.1f'  ) ' %' ];
    legends{4} = [ 'Estate: ' num2str( props(4),'%7.1f' ) ' %' ];
    legends{5} = [ 'Fees: '    num2str( props(5),'%7.1f' ) ' %' ];
```

```
% create chart
 if min( props ) >= 0
  % create a pie chart
   if min( props )>0.05
        labels = { client.p1Name, client.p2Name, 'Both', 'Estate', 'Fees' };
     else
       labels = { '', '', '', '', '' };
   end;
    h = pie( props, labels );
   set( h( 2:2:10 ), 'FontSize', 20 );
  % create legend
   legends = {  };
   legends{1} = [client.p1Name ': ' num2str(props(1), '%7.1f ' ) ' %'];
   legends{2} = [client.p2Name ': ' num2str(props(2), '%7.1f ' ) ' %'];
   legends{3} = ['Both: '   num2str(props(3), '%7.1f ' ) ' %'];
   legends{4} = ['Estate: ' num2str(props(4), '%7.1f ' ) ' %'];
   legends{5} = ['Fees: '    num2str(props(5), '%7.1f ') ' %'];
   legend( legends, 'Location', 'SouthEastOutside' );
   cmap = [ 1 0 0; 0 0 1; 0 .8 0; 1 .5 0; 0 0 0 ]; colormap( gcf, cmap );
  else
  % create a bar chart
   bar( props );
   grid;
   ylabel( 'Percent of Total Value' );
   labels = { client.p1Name, client.p2Name, 'Both', 'Estate', 'Fees' };
   set( gca, 'XTickLabel', labels );
 end; % if min(props) >= 0

 % add title
  title2 = [ 'Total Value = $'  totalValStg ' thousand'];
  title( {'Recipient Present Values',title2}, 'color', [0 0 1] );

end % function recipientPVs
```

Matlab's pie chart function is a bit fussy and dislikes producing wedges representing zero percent of the total. To placate it, we change any zero value to a very small positive number. Also, in the unlikely event that any proportion of value is negative, we completely give up the attempt to produce a pie chart. providing a bar chart instead. Of course this should not happen, but better safe than sorry. To minimize the possibility of labels overlapping one another, we also require that every portion constitute at least 5% of the total before we attach labels to its wedge of the pie.

The rest of the statements in the function are necessary but not especially exciting. We leave exploration of their properties it to diligent programming aficionados.

PPCs and Incomes

In chapter 8, we showed that the most cost-efficient way to obtain a given distribution of incomes in a year is to arrange for larger real incomes to be obtained in cheaper states. More precisely, real income should be a non-increasing function of present value (and therefore also of price per chance).

To see whether this is the case, and the effects of insuring that it is, we add an animated graph with PPCs and real incomes to our analytic library. The commands included in the *analysis_create* function are:

```
% plot PPCs and Incomes -- y/n
    analysis.plotPPCSandIncomes = 'n';
% plot PPC and Incomes --  semilog or loglog
    analysis.plotPPCSandIncomesSemilog = 'y';
% plot PPCs and Incomes -- sets of states (one set per graph)
    analysis.plotPPCSandIncomesStates = { [3] };
% plot PPCs and Incomes: minimum percent of scenarios
    analysis.plotPPCSandIncomesMinPctScenarios = 0.5;
```

The second element indicates whether the graph is to plot the logarithm of PPC on the vertical axis and income on the horizontal (giving a *semilog* graph) or to plot the logarithms of both PPC and Income (resulting in a *loglog* graph). In either case,we use logarithms for PPC since the range of possible values is very large and all possible values are positive. The default approach is to plot income on the horizontal axis since this is more familiar and can accommodate zero values of income. The alternative version is included for situations in which it is important to reflect the client's implied relative risk aversion, which is shown by the slope of a curve or line in a loglog plot. For cases in which the minimum income is zero, the semilog version will be used automatically since the logarithm of zero is minus infinity.

The third element indicates which states are to be included in each graph, following our usual convention, with a separate plot produced for each vector in the cell array. The final element provides a cutoff, so that only years in which there are sufficient scenarios (here, at least 0.5% of the total number of scenarios) are shown.

The statements added to the *analysis_process* function are straightforward:

```
% analysis: plot PPCs and Incomes
  if analysis.plotPPCSandIncomes == 'y'
    % find states;
     states = analysis.plotPPCSandIncomesStates;
    % create figures
     for i = 1:length( states )
       % create Figure
         analysis = createFigure( analysis, client );
       % call external function analPPCSandIncomes
          analPlotPPCSandIncomes (analysis, client, market, states{i} );
       % process figure
          analysis = processFigure( analysis );
      end; % i
  end; % if analysis.plotPPCSandIncomes == 'y'
```

No surprises here.

The work is done in a long external function named *analPlotPPCSandIncomes*. Many of the statements will be familiar. Here is the function:

```
function analPlotPPCSandIncomes( analysis, client, market, states );
  % plot PPCS and incomes for states
  % called by analysis_process function

% add labels
   set( gcf, 'name', ['PPCs and Real Incomes '] );
   set( gcf, 'Position', analysis.figPosition );

   grid on;
   ylabel(  'log ( Price per Chance )  ' );
   hold on;

 % 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 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 ];

 % get matrix size
   [nscen nyrs] = size( client.pStatesM );

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

 % set initial delay
   delay = delays(1);

 % 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;
```

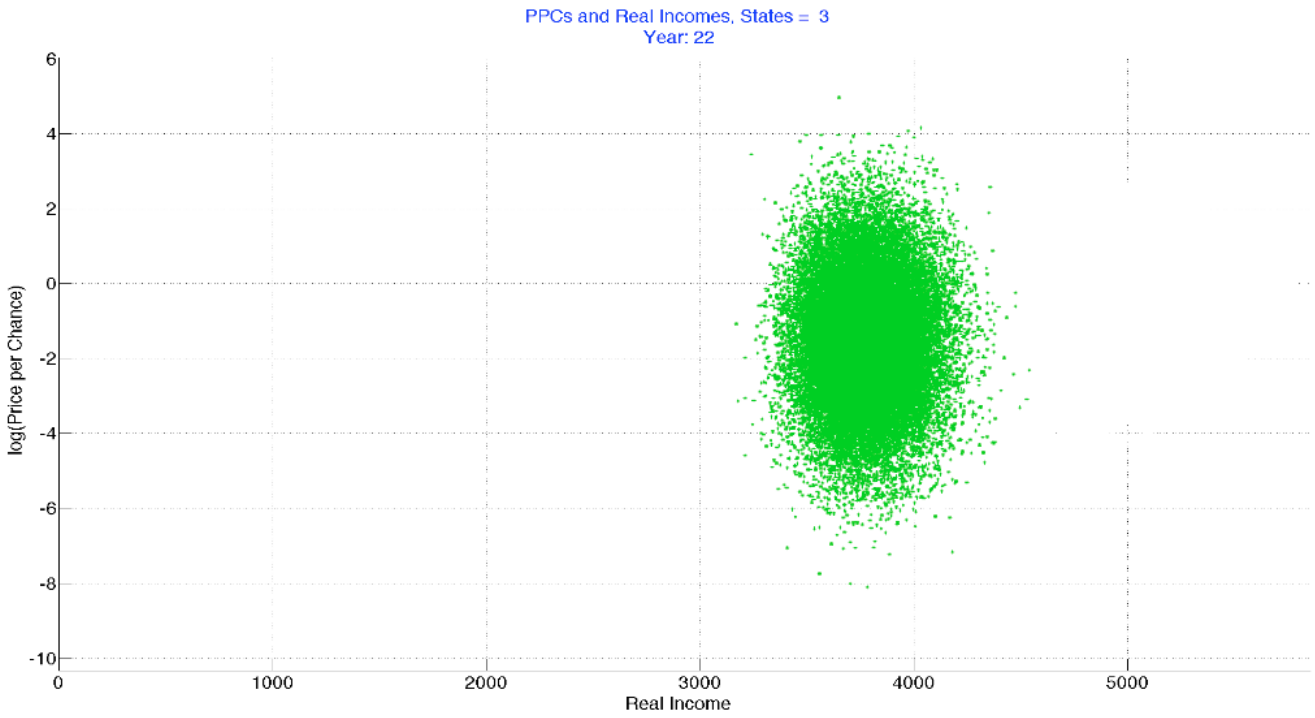
```
% find last year with sufficient included states
  [nscen,nyrs] = size( cells );
  numstates = sum( cells > 0 );
  minprop = analysis.plotPPCSandIncomesMinPctScenarios;
  minnum = ( minprop / 100 ) * nscen;
  lastyear = max( ( numstates > minnum ).*( 1:1:nyrs ) );
  if lastyear == 0
     title( 'Insufficient scenarios' );
     return
  end;

% truncate matrices
  cellsM = cells( :, 1:lastyear );
  incsM = client.incomesM( :, 1:lastyear );
  ppcsM = market.ppcsM( :, 1:lastyear );

% find maximum and minimum incomes
   ii = find( cellsM > 0 );
   incsvec = incsM( ii );
   maxinc = max( incsvec );
   mininc= min( incsvec );
% find maximum and minimum PPCs
   ppcsvec = ppcsM( ii );
   maxppc = max( ppcsvec );
   minppc = min( ppcsvec );

% if minimum income is zero, require semilog
  if mininc == 0
     analysis.plotPPCSandIncomesSemilog = 'y';
  end;
```

```
if analysis.plotPPCSandIncomesSemilog == 'y'

 % set axes and label
   axis( [ 0 maxinc log(minppc) log(maxppc) ] );
   xlabel( 'Real Income  ' );

  for yr = 1 : lastyear

   % get data
    cellsv = cellsM( :, yr );
    ii = find( cellsv > 0 );
    incs = incsM( ii, yr );
    ppcs = ppcsM( ii, yr );

   % title
    ttl1 = [ 'PPCs and Real Incomes, States =  '  num2str(states) '  ' ];
    ttl2 = [ 'Year: ' num2str(yr) '  ' ];
    title( {ttl1 ttl2}, 'color', 'b' );

   % plot points
    plot( incs, log(ppcs), '*', 'color', clrFull, 'Linewidth', .5 );
    pause( delay );
   % shade points
    delay = delay + delayChange;
    plot( incs, log( ppcs ), '*', 'color', clrShade, 'Linewidth', .5 );

  end;

end; % if analysis.PlotPPCSandIncomesSemilog == 'y'
```

```
if analysis.plotPPCSandIncomesSemilog ~= 'y'

 %  set axes and labels
   xlabel( 'log ( Real Income )  ' );
   axis( [ log(mininc) log(maxinc) log(minppc) log(maxppc) ] );

  for yr = 1 : lastyear

   % get data
    cellsv = cellsM( :, yr );
    ii = find( cellsv > 0 );
    incs = incsM( ii, yr );
    ppcs = ppcsM( ii, yr );

   % title
    ttl1 = [ 'PPCs and Real Incomes, States =  '  num2str(states)  '  ' ];
    ttl2 = [ 'Year: ' num2str(yr) '  ' ];
    title( {ttl1 ttl2}, 'color', 'b' );

   % plot points
    plot( log(incs), log(ppcs), '*', 'color', clrFull, 'Linewidth', .5 );
    pause( delay );
   % shade points
    delay = delay + delayChange;
    plot( log(incs), log(ppcs), '*', 'color', clrShade, 'Linewidth', .5 );

  end;

 end; % if analysis.PlotPPCSandIncomesSemilog ~= 'y'


end % plotPPCSandIncomes(analysis, client,market, states);
```

Note that we plot the logarithms of PPCs on the y-axis due to the large range of their values. However, due to the possible inclusion of incomes with zero values we provide for a linear scale for the x-axis if needed. And, to give a sense of relative values, in such a case we set the origin for that axis to zero.

Here is a “still picture” from an analysis of our fixed nominal annuity for personal state 3, taken when year 22 was being shown.



As can be seen, the real incomes in each year varied due to the impact on a fixed nominal payment of differing rates of cumulative inflation. But we assume no correlation between inflation and the real rates of return on the market. And, since both present values and PPC values are related to cumulative market real returns, there is no correlation between the real annuity values and PPCs, as the points in the graph show. A similar situation holds for every year after the initial period.

Clearly, this is not a cost-efficient strategy, since real income is not a one-to-one function of PPC. To be efficient in this sense, one should not receive higher incomes in more expensive states. If this were in fact a cost-efficient strategy, the points in the graph would fall along a curve that is everywhere vertical or downward-sloping, and the curve could be interpreted as showing the implied marginal utility of income for the year and personal states covered. But no such interpretation is possible in this case.

Yearly Present Values

The next type of analysis provides information concerning the present value of possible incomes in each future year; it also provides a measure of the cost efficiency for each year as well as a measure for the entire matrix of possible future scenarios.

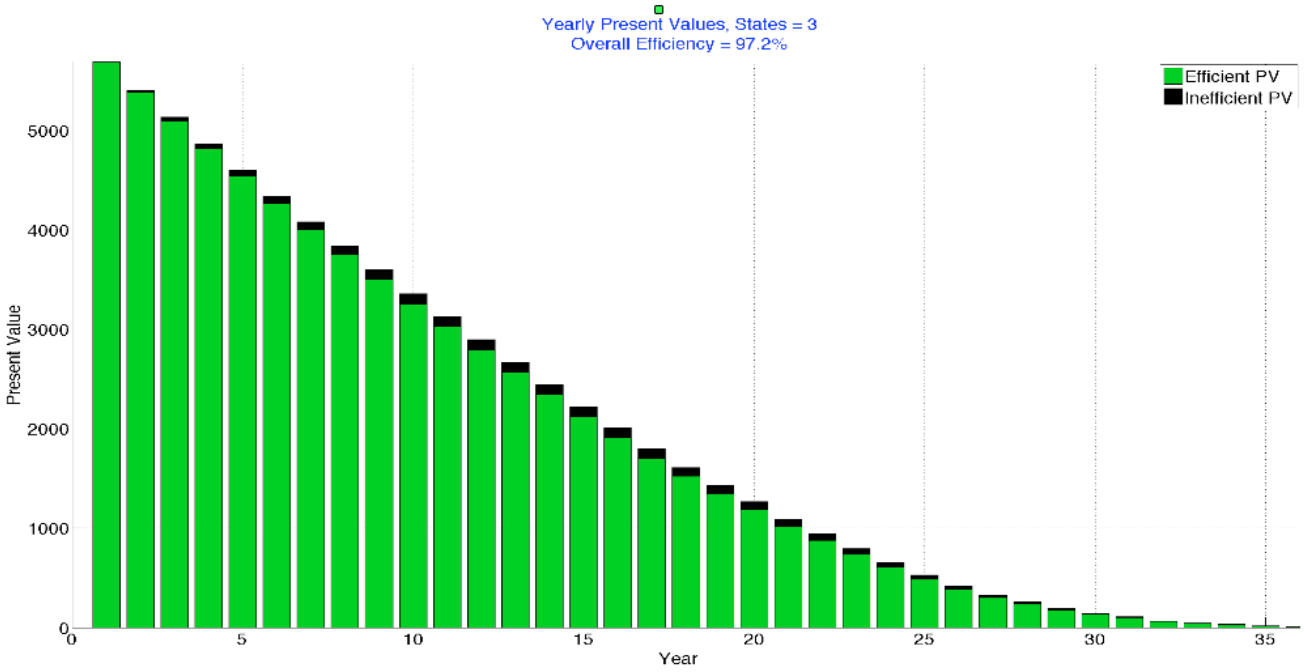
The key calculations required to compute such measures are straightforward. Here they are:

```
pvs  = market.pvsM( rows, yr );
incs = client.incomesM( rows, yr );
totalpv = pvs' * incs;
effpv = sort( pvs, 'ascend' )' * sort( incs, 'descend' );
```

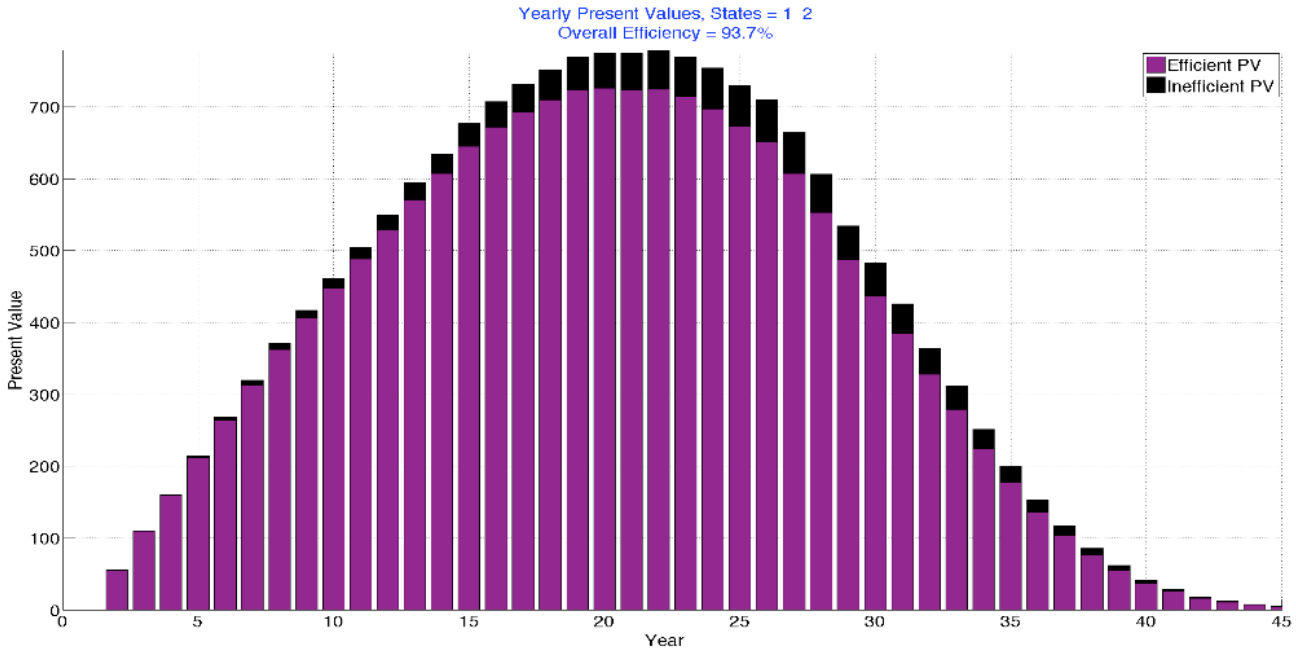
The vector *rows* has the numbers of rows that are relevant for the personal states being analyzed in a specific year (*yr*) . The total present value of the incomes in that year for the selected states (*totalpv*) is obtained by multiplying the present value for each cell by the income in that cell, then summing the results. The cost-efficient counterpart creates a vector of present values in ascending order and another of incomes in descending order, multiplies their elements and adds their products. This guarantees that if one income is larger than another it will be assigned to a scenario with a present value with a lower or equal value. Summing the products gives the efficient present value (*effpv*): the lowest possible cost for which the selected set of incomes could be provided.

The figure below shows results for personal states in which both Bob and Sue are alive, with income provided by our fixed nominal annuity. The total height of each bar is the present value of the incomes for the indicated year. The height of the each green bar is the present value associated with a cost-efficient strategy that would provide the same probability distribution of incomes in the year in question. The difference (shown by the black segment at the top of each bar) is the amount that could be saved by adopting an efficient strategy. The percentage shown in the title indicates the ratio of the total area of the green bars to the total area of the bars (including both green and black portions).

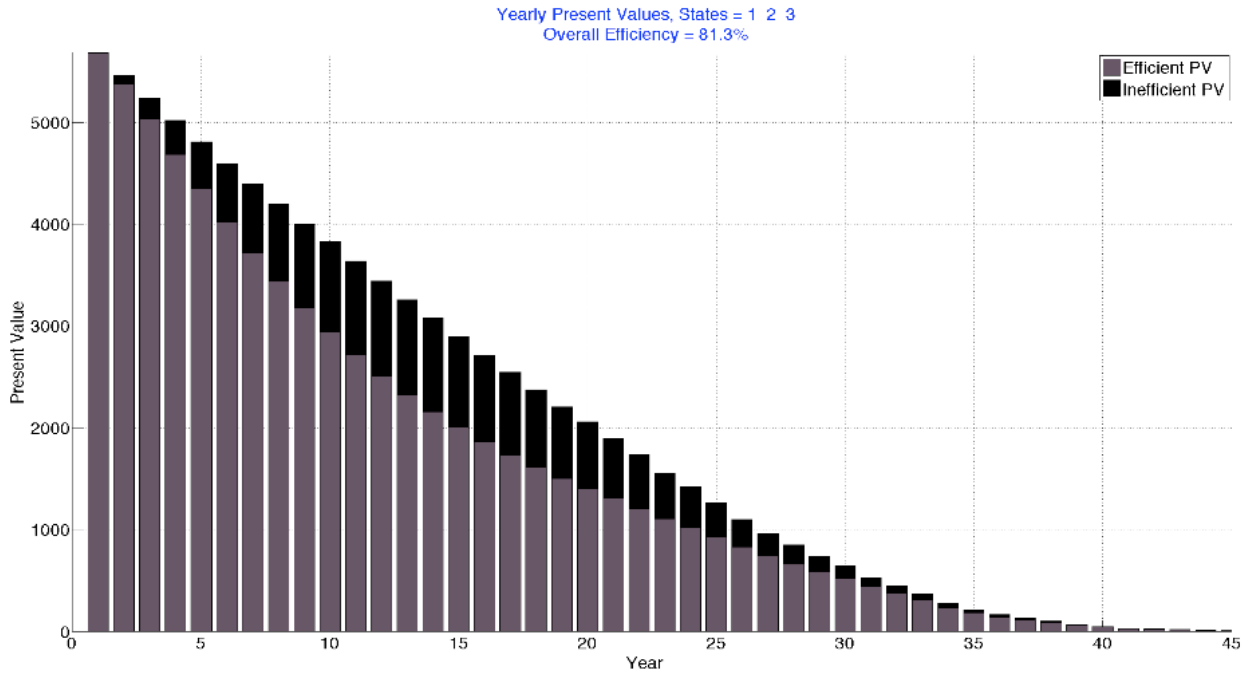
In this case, the inefficiency caused by the uncertainty due to factors other than the return on the market portfolio (here, inflation) results in a set of possible incomes that is only worth 97.2% of the amount paid. In other words, the same probability distribution of incomes could be obtained for 97.2% of the total cost by taking only market risk. We will have more to say about this in the next section.



Here is the analysis for personal states 1 and 2, when only Bob or Sue is alive. Note that the efficiency is considerably lower (93.7%). This is due to the fact that the associated personal states are more likely to occur in later years when the effects of uncompensated uncertainty in real returns due to inflation are greater.



When performing this sort of analysis it is important that only meaningful personal states be included. Here is an example of a very bad choice for our nominal fixed annuity with 50% joint and survivor benefits.



The problem is not hard to identify. The annuity pays half as much in states 1 and 2 as it does in state 3. By including all three states in one analysis, we allow the efficiency calculation to happily assign incomes in states 1 or 2 to higher-present value states and incomes in state 3 to lower-present value states. This is clearly not feasible, so the results make no economic sense. We will have more to say about this shortly.

Now to the programs that create yearly present values graphs.

First, we add the following to the *analysis_create* function:

```
% plot yearly present values -- y (yes) or n (no)
    analysis.plotYearlyPVs = 'n';
% plot yearly present values-- sets of states (one set per graph)
    analysis.plotYearlyPVsStates = { [3] [1 2]  };
% plot yearly present values: minimum percent of scenarios
    analysis.plotYearlyPVsMinPctScenarios = 0.5;
```

Next, we add these statements to the *analysis_process* function:

```
% analysis: plot yearly PVs
    if analysis.plotYearlyPVs == 'y'
      % find states;
       states = analysis.plotYearlyPVsStates;
      % create figures
        for i = 1 : length( states )
          % create Figure
            analysis = createFigure( analysis, client );
          % call external function analPlotPPCSandIncomes
             analPlotYearlyPVs( analysis, client, market, states{i} );
          % process figure
             analysis = processFigure( analysis );
        end; %i
    end; % if analysis.plotYearlyPVs == 'y'
```

Finally, the tedious external function *analPlotYearlyPVs,* which uses the code we saw earlier in this section as well as other approaches that we have seen in previous sections of this chapter.

```
function analPlotYearlyPVs( analysis,  client,  market,  states );
  % plot Yearly PVs for states
  % called by analysis_process function

 % add labels
   set( gcf, 'name', 'YearlyPVs   ' );
   set( gcf, 'Position',  analysis.figPosition );

   grid on;
   xlabel( 'Year   ' );
   ylabel( 'Present Value    ' );
   hold on;

 % 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 efficient PV color based on states
     clrmat = [ ];
     for s = 1:length( states )
       clrmat = [ clrmat; cmap(states(s)+1, :) ];
     end;
     clrPV = mean( clrmat, 1 );
 % set inefficiency color
     clrIneff = [ 0 0 0 ];
     colormap( [ clrPV ; clrIneff ] );

 % get matrix size
   [nscen nyrs] = size( client.pStatesM );

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

 % set initial delay
   delay = delays(1);

 % 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 sufficient included states
 [nscen,nyrs] = size(cells);
 numstates = sum( cells > 0 );
 minprop = analysis.plotYearlyPVsMinPctScenarios;
 minnum = ( minprop / 100 )*nscen;
 lastyear = max( (numstates > minnum) .* (1:1:nyrs) );
 if lastyear == 0
    title('Insufficient scenarios');
    return
 end;

% truncate matrices
 cellsM = cells( :, 1:lastyear );
 incsM = client.incomesM( :, 1:lastyear );
 ppcsM = market.ppcsM( :, 1:lastyear );

% set up valuation vectors
 totalpvs = [ ];
 effpvs = [ ];
% get present values
 for yr = 1:lastyear
    rows = find( cells(:,yr) > 0 );
    pvs = market.pvsM( rows, yr );
    incs = client.incomesM (rows, yr );
    totalpv = pvs' * incs;
    effpv = sort( pvs, 'ascend' )' * sort( incs, 'descend' );
    totalpvs = [ totalpvs totalpv ];
    effpvs = [ effpvs effpv ];
 end;

% compute total efficiency
 totaleff =100*( sum(effpvs) / sum(totalpvs) );

% title
  ttl1 = [ 'Yearly Present Values, States = ' num2str(states)  ' '  ];
  ttl2 =[ 'Overall Efficiency = ' num2str( .1*round(10*totaleff) )  '%  ' ];
  title( {ttl1 ttl2}, 'color' , 'b' );

% scale axes
  axis( [ 0 lastyear+1 0 max(totalpvs) ] );
  grid;

% plot pvs
  diffs = totalpvs - effpvs;
  bar( [effpvs; diffs]',  'stacked' );  grid;
  legend( 'Efficient PV ', 'Inefficient PV ' );

end % plotYearlyPVs(analysis, client,market, states);
```

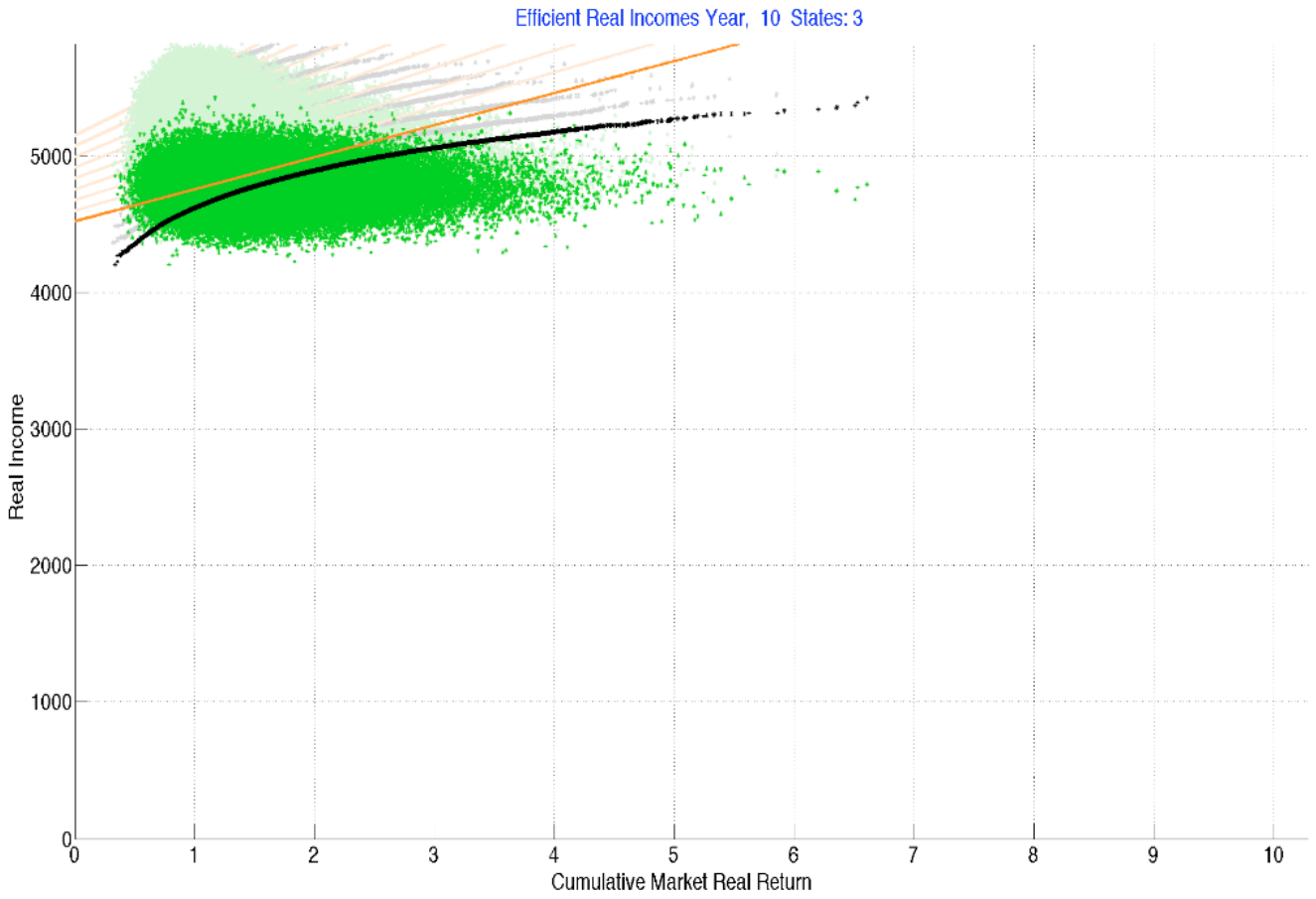
Efficient Incomes

Our final type of analysis combines aspects of the PPCs and Incomes analysis with some aspects of Yearly Present Values. We know that the least-cost way to provide a probability distribution of real incomes in a given year is to have the relationship between real income and price per chance be monotonic and non-increasing. In other words: the more the real income, the cheaper the price per chance. And, since each scenario is equally likely, the more the real income in a scenario, the lower the present value of $1 of income in that scenario.

Our equilibrium model of security prices holds that for any given future year, the present value of $1 in a scenario will be a non-increasing function of the cumulative market return up to that year. More simply, the greater the cumulative return on the market in a scenario, the lower will be the present value of $1 of income in that scenario.

Combining these two relationships leads to the conclusion that to obtain a given distribution of real incomes across scenarios for a given year for the lowest cost, we should arrange to have lower incomes in scenarios in which the market return is lower and higher incomes in scenarios in which the market return is higher). More precisely, for the lowest-cost strategy for any given year, real income should be a non-decreasing function of cumulative market return up to that year.

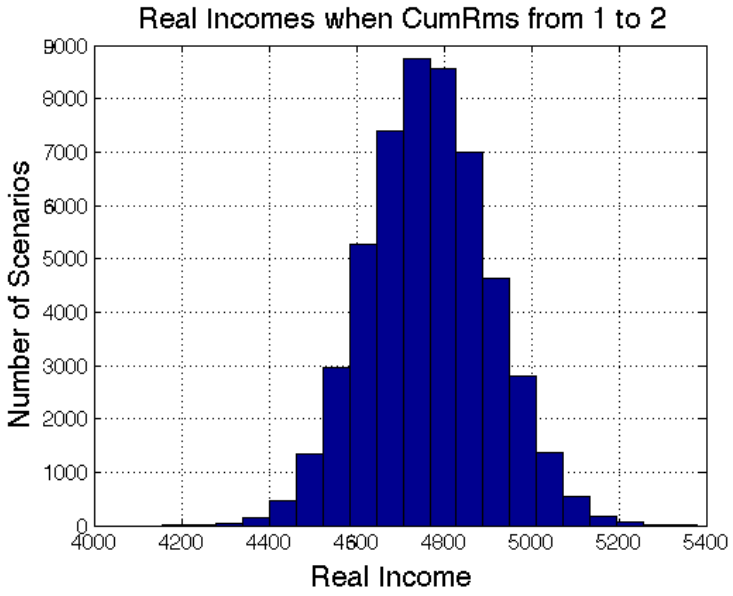
The *PlotEfficientIncomes* analysis shows the results of pursuing and achieving cost-efficiency in each year. The figure below shows the results obtained for year 10 with our fixed nominal annuity, focusing on the incomes obtained when both Bob and Sue are alive (personal state 3).

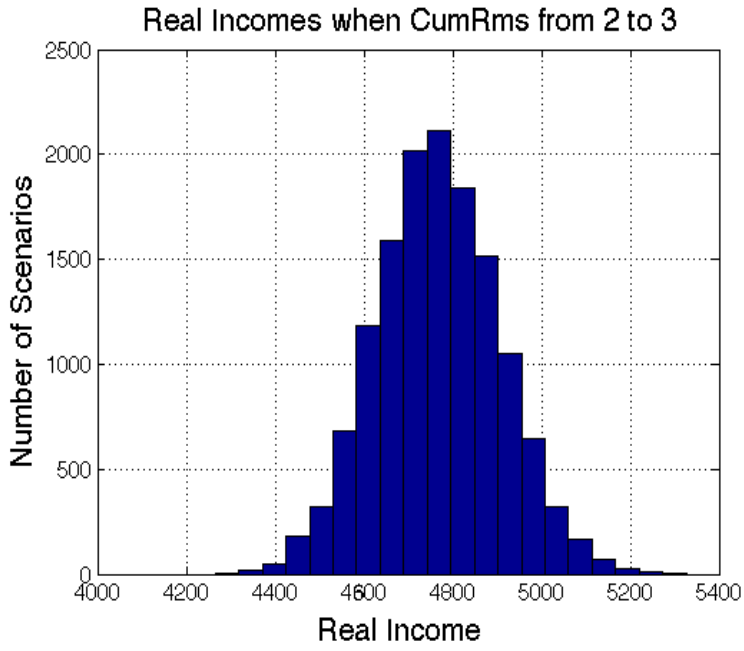


The horizontal axis plots *cumulative market return.* The vertical axis shows real income in the year in question.

The green dots represent the real incomes obtained in each of the scenarios for the chosen personal state or states. As we know, these will vary due to the impact of inflation on real value the fixed nominal income. However, since we assume that inflation is independent of the real return on the market, there is no pervasive relationship between real income and market return – a regression line fit to the green dots would almost certainly be horizontal.

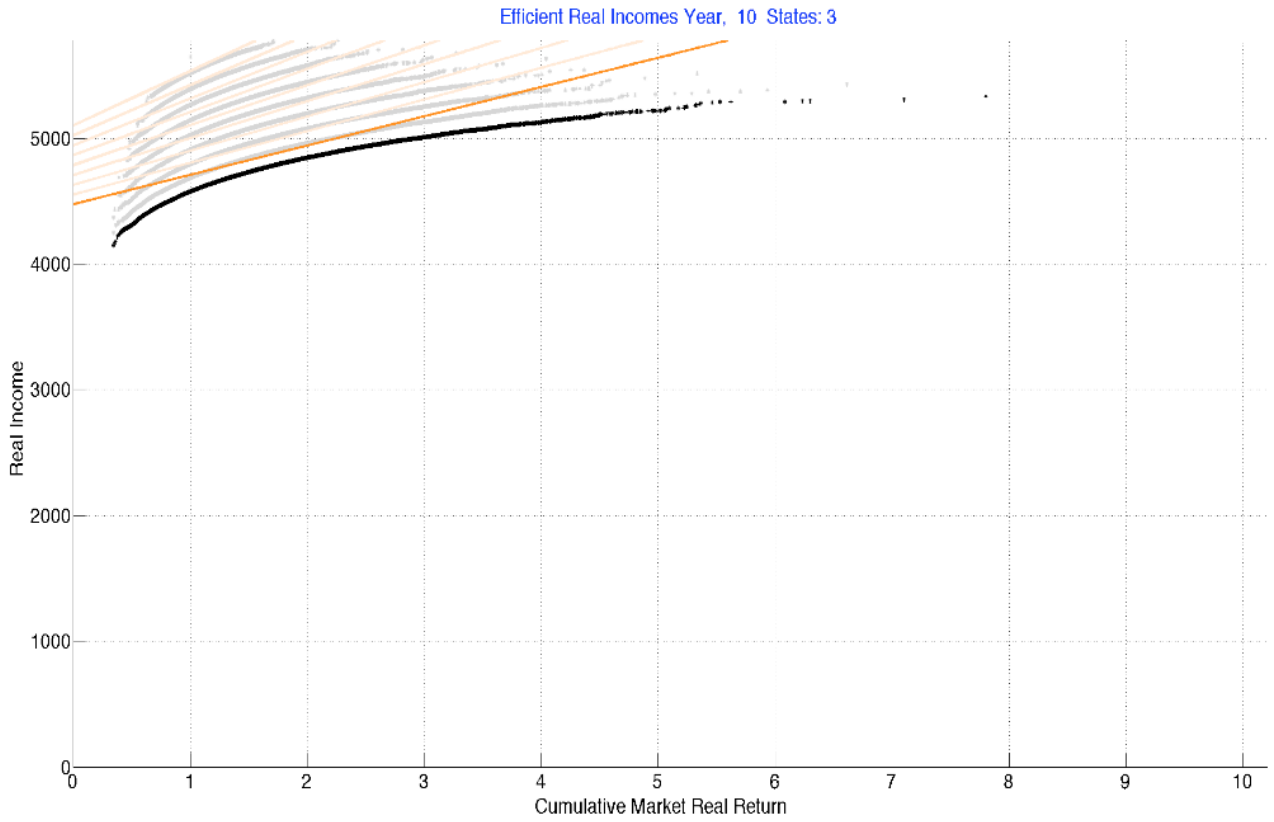
One aspect of this plot is unfortunate, to say the least. Visually, the vertical scatter of the points appears to become smaller as one moves to the right. This seems to imply that for this retirement income strategy, there is less uncertainty about real income in scenarios with high cumulative market returns than in scenarios with low such returns. In fact this is not the case. Recall that in this case, differences in real income in a given year are caused entirely by differences in cumulative inflation, and we assume that there is no correlation between inflation and the real return on the market in any year. Hence, for a given year, the range of possible real incomes in scenarios with a given cumulative market return should be similar to that for the scenarios with any other cumulative market return. And this is indeed the case, as shown by the following histograms of the distributions of real returns for scenarios in two different ranges of cumulative market returns.





While the histograms are similar, the scales are very different, since there are fewer scenarios in the second range. How then, to explain the scatter of points in our efficient real incomes graph? Compare, for example the points for the scenarios in which the cumulative market return in year 10 is 2.0 with those for which it is 3.0. The vertical scatter appears to be much greater for the former than for the latter. But in fact the actual distributions are similar. The problem is that there are more points in the former case and many of them plot at the same screen pixel. If the color of each pixel were proportional to the number of scenarios plotting in its associated range of values the picture would be more informative. But this is not the case; hence the confusion.

As we will see, an alternative is available. An analysis data element allows for plots with any desired combination of points, curves and lines. Here, for example, is the figure for our nominal fixed annuity for year 10 and personal state 3 without the green points.



The points on the black curve show the real incomes and cumulative market returns for a cost-efficient strategy that would provide precisely the same probability distribution as does the fixed nominal annuity. By design, the points plot as a non-decreasing function of cumulative market return. And, as we know, the cost of such a strategy will be equal to or less than that of

the actual strategy (green dots).

The other feature of the diagram is an orange line for each future year. As we know, a straight line in such a graph shows incomes obtained from a strategy that invests an initial amount in the riskless asset and another amount in the market portfolio. Each such line is constructed in two steps. First, a straight line is fit to the points on the black curve, using linear regression. Then the line is moved upward or downward, holding the slope constant, until the present value of the plotted incomes equals that of the original set of incomes. The result is a *two-asset market-based strategy* with properties similar to those of the original approach and the same cost. As can be seen in the earlier graph, this strategy provides more income in many scenarios (those with green points below the line) but less income in others (those with green points above the line).

Here are the key attributes of this analysis of our fixed nominal annuity. For any given year, the costs of the actual strategy (green dots) and the two-asset market-based strategy (orange line) are the same, but the cost of the cost-efficient equivalent (black curve) is less. The probability distributions of real income are the same for the actual strategy and the cost-efficient strategy, but the probability distribution of real income for the two-asset market-based strategy (orange line) is better, since for most years it lies everywhere above the black curve.

The incomes shown by the set of orange lines could be provided by an insurance company that invested in a series of *lockboxes*, one to provide income for survivors in each future year. Each lockbox would have an appropriate combination of the riskless real asset (Tips) and the market portfolio, with the proceeds to be used to make annuity payments to those surviving until the year in its designated year. Chapter 16 discusses such lockbox annuities at length, in the hope that some insurance company might create such policies in the future.

One last task remains for this chapter: to include the program statements required to create graphs such as these and comment briefly about any novel aspects. As usual, only those fascinated with computer programs need continue. Others may move to the next chapter that, mercifully, has mostly words and graphs.

We begin our discussion of programs, as usual, with statements added to the *analysis_create* function:

```
% plot efficient incomes -- y (yes) or n (no)
    analysis.plotEfficientIncomes = 'n';
 % plot efficient incomes -- sets of states (one set per graph)
    analysis.plotEfficientIncomesStates = { [3]  [1 2] };
 % plot points (actual) curves (efficient) and/or
 %   lines (two-asset market-based strategies):
 %   combinations of (p ,c, l) -- one graph per type
    analysis.plotEfficientIncomesTypes = { 'pcl' };
 % plot efficient incomes: minimum percent of scenarios
    analysis.plotEfficientIncomesMinPctScenarios = 0.5;
```

The second statement creates a cell array with the state or states to be selected for each plot. As discussed earlier, only states in which rearranging incomes in different scenarios is feasible should be used for each such analysis. The default is to use state 3 for one plot and combine states 1 and 2 for another.

The next data element specifies the sets of incomes to be shown in each of one or more plots. A single graph can show: points for the actual real incomes (*p*) , curves for efficient combinations (*c*) and/or lines for two-asset market-based strategies (*l*).

The final element is similar to that used for some other analyses. It restricts the analysis to years in which there is income in a sufficiently large percent of scenarios, in order to avoid plots with relatively few data points.

Since each graph includes detailed information for many years, animation is used, with each year's data plotted in dark colors then, after a delay, replotted in lighter shades before the next year's data are shown. As in every such case, the delay times and degree of shading are determined by the corresponding elements in the analysis data structure.

Here are the statements added to the *analysis_process* function:

```
% analysis: plot efficient incomes
    if analysis.plotEfficientIncomes == 'y'
      % find states;
       states = analysis.plotEfficientIncomesStates;
      % find types
       types = analysis.plotEfficientIncomesTypes;
      % create figures
        for i = 1 : length( types )
         for j = 1 : length( states )
          % create Figure
            analysis = createFigure( analysis, client );
          % call external function analPlotPPCSandIncomes
            analPlotEfficientIncomes( analysis, client, market, types{i}, states{j} );
         % process figure
            analysis = processFigure( analysis );
        end; %j
       end; %i
    end; % if analysis.plotEfficientIncomes == 'y'
```

Nothing notably new here. The hard work is done by an external function, in this case *analPlotEfficientIncomes*. As in other cases in which multiple states and/or types may be required, a separate graph is provided for each possible combination of a state and type.

Now for *analPlotEfficientIncomes,* the long and tedious function that does the hard work.

```
function analPlotEfficientIncomes( analysis, client, market, type, states );
  % plot efficient incomes for states
  % called by analysis_process function

% add labels
    set( gcf, 'name', 'Efficient Real Incomes ' );
    set( gcf, 'Position', analysis.figPosition );

    grid on;
    xlabel(  'Cumulative Market Real Return   '  );
    ylabel( ' Real Income     ' );
    hold on;

 % set colors for points 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 point color based on states
     clrmat = [ ];
     for s = 1 : length( states )
       clrmat = [ clrmat; cmap( states(s)+1, : ) ];
     end;
     clrPoints = mean( clrmat, 1 );

 % set point shadow shade color
     shade = analysis.animationShadowShade;
     clrPointsShade = shade*clrPoints + ( 1-shade )*[ 1 1 1 ];

 % set curve color and shade color
     clrCurve = [ 0 0 0 ];
     clrCurveShade = shade*clrCurve + ( 1-shade )*[ 1 1 1 ];

 % set line color and shade color
     clrLine = [ 1 0.5 0 ];
     clrLineShade = shade*clrLine + ( 1-shade )*[ 1 1 1 ];

 % 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 sufficient included states
   [nscen, nyrs] = size( cells );
   numstates = sum( cells > 0 );
   minprop = analysis.plotEfficientIncomesMinPctScenarios;
   minnum = ( minprop / 100 ) * nscen;
   lastyear = max( (numstates > minnum) .* ( 1:1:nyrs ) );
   if lastyear == 0
      title( 'Insufficient scenarios' );
      return
   end;

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

  % truncate matrices
   cellsM = cells( :, 1:lastyear );
   incsM = client.incomesM( :, 1:lastyear );
   cumretsM = market.cumRmsM( :, 1:lastyear );
   pvsM = market.pvsM( :, 1:lastyear );

  % find maximum incomes
   maxincs =  max( max( incsM.*cellsM ) );

  % find maximum cumulative market return for x axis
  %   includes 99.9% of possible values
   cumretm = cumretsM .* cellsM;
   cumretv = sort( cumretm(:), 'ascend' );
   maxcumrets = cumretv( 0.999*length(cumretv) );

  % scale axes
   axis( [ 0 maxcumrets 0 maxincs ] );
   grid on;
```

```
% plot results
 for yr = 1 : lastyear

  % get data for year
   rows = find( cells(:,yr) > 0 );
   pvs = pvsM( rows, yr );
   incs = incsM( rows, yr );
   cumrets = cumretsM( rows, yr );

  % sort data
   cumretsS = sort( cumrets, 'ascend' );
   incsS = sort( incs, 'ascend' );
   pvsS  = sort( pvs, 'descend' );

  % plot points if desired
   if length( findstr( 'p', type ) )  > 0
     plot( cumrets, incs, '*', 'color', clrPoints );
   end;

  % fit line for regression of sorted incomes and cumrets
  %   incomeS = b(1) + b(2)*cumretS
   xvals =  [ ones( length(cumretsS), 1)  cumretsS ];
   b = xvals \ incsS;

  % compute fitted incomes using regression equation
   incsFitted = b(1) + b(2)* cumretsS;
  % compute present value of original set of incomes
   pvIncs =  sum( pvs .* incs );
  % compute present value of fitted line
   pvLine =  sum( sort(pvs, 'descend' ) .* sort(incsFitted, 'ascend' ) );
  % find additional income for each scenario
   delta =  ( pvIncs – pvLine ) / sum( pvs );
  % increase each fitted income by a constant so pv = orginal amount
    incsFitted = incsFitted + delta;

  % plot sorted cumrets and incomes if desired
   if length( findstr( 'c', type ) ) >0
     plot( cumretsS, incsS, '*', 'color', clrCurve );
   end;

  % plot fitted line if desired
   if length( findstr( 'l', type ) ) >0 & ( yr > 1 )
      plot( [0;cumretsS], [b(1)+delta; incsFitted], 'color',  clrLine, 'Linewidth', 4 );
   end;
```

```
    % add title
     ttl1 = [ 'Efficient Real Incomes Year,  ' num2str(yr) '  States: ' num2str(states) ];
     title(ttl1, 'color', 'b');

    % pause
     pause( delay );

    % shade points if plotted
     if length( findstr('p',type) ) > 0
      plot( cumrets, incs, '*', 'color', clrPointsShade );
     end;

    % shade sorted cumrets and incomes if plotted
     if length( findstr('c', type) ) > 0
      plot( cumretsS, incsS, '*', 'color', clrCurveShade );
     end;

    % shade fitted line if plotted
     if length( findstr('l', type)) > 0 & ( yr > 1 )
       plot( [0;cumretsS], [b(1)+delta; incsFitted], 'color',  clrLineShade, 'Linewidth', 4 );
     end;

    % pause
     pause( delay );

   % change delay time
     delay = delay - delayChange;

  end;

end % analPlotEfficientIncomes(analysis, client,market, types, states);
```

Many of these statements are similar to those used in other analysis functions and need no elaboration here. One novelty is the choice of an upper bound for the horizontal axis that plots cumulative market returns. To avoid the chance of an extremely large such return extending the axis so far to the right that the vast majority of the information is squeezed to the left, we plot only results up to the 99.9'th percentile of all cumulative market returns. This omits a few points but provides a far better visualization for the rest.

A novel aspect of the function is the construction of the orange line for each year showing the incomes from a two-asset market-based strategy. We start by fitting a straight line to the sorted incomes and cumulative market returns. The equation we seek to fit is:

$$y = b(1) + b(2) * x + e$$

where x is a vector of our sorted cumulative market returns, y is a vector of our sorted incomes, and e is an error term. The goal is to find values for the parameters b(1) and b(2) that will give the lowest possible value for the the standard deviation of the error terms. Here are the statements that do the job:

```
% fit line for regression of sorted incomes and cumrets
  %   incomeS = b(1) + b(2)*cumretS
    xvals =  [ ones( length(cumretsS), 1  cumretsS ];
    b = xvals \ incsS;
```

First we create a matrix *xvals* with *ones* in the first column and the values of the independent variable (sorted cumulative returns) in the second column. Then we use matlab's backslash operator with this matrix as the first argument and the vector of dependent variables (*incS*) as the second argument. The result is a vector with two *regression coefficients* b(1) and b(2). The regression line is thus

$$incFitted = b(1) + b(2) * cumretsS$$

Next we compute the fitted incomes:

```
% compute fitted incomes using regression equation
    incsFitted = b(1) + b(2)* cumretsS;
```

and then use the present values to find the present value of the original set of incomes and that of the fitted set of incomes along the regression line:

```
% compute present value of original set of incomes
  pvIncs =  sum( pvs .* incs );
% compute present value of fitted line
  pvLine =  sum( sort(pvs, 'descend' ) .* sort(incsFitted, 'ascend' ) );
```

It remains to compute *delta*, the difference in each of the fitted incomes that will make the present value of all of them equal to that of the original set of incomes, then this amount to each of the fitted values to obtain a new set of incomes that will plot on a line parallel to the regression line but cost as much as the original set of incomes:

```
% find additional income for each scenario
  delta =  ( pvIncs – pvLine ) / sum(pvs);
% increase each fitted income by a constant so pv = orginal amount
  incsFitted = incsFitted + delta;
```

Finally, if requested the results are plotted as an orange line.

Script File and Video

Most of the analyses in this chapter can be produced by the script file *SmithCase_Chapter13.m*, which is included in the directory containing the RISMAT functions. A video the output produced with that script is available at:

www.stanford.edu/~wfsharpe/RISMAT/SmithCase_Chapter13.mp4

This completes our descriptions of programs designed to analyze aspects of incomes, fees and values. The next several chapters introduce a number of additional possible strategies for providing retirement incomes, provide programs to compute matrices of incomes and fees produced by such strategies, and utilize our analytic tools to find their salient properties.