mkmod5 Code Review

Status

November 2010

Jim Brannon, Leonard Rice Engineers

Overview

- **mkmod**[1,2,3,4,5] is a data processing tool that creates a MODFLOW (ESPAM) input data set using data stored in the several and various data files created to describe the state over time of the natural and human systems affecting the aquifer.
- written in the computer programming language PERL and has evolved over the past 5+ years to over 1300 lines of code
- the PERL code should instantiate in logic and data the "conceptual models" of the behavior of the components of the ESPAM system that the IDWR and ESHMC have agreed upon

Overview (cont)

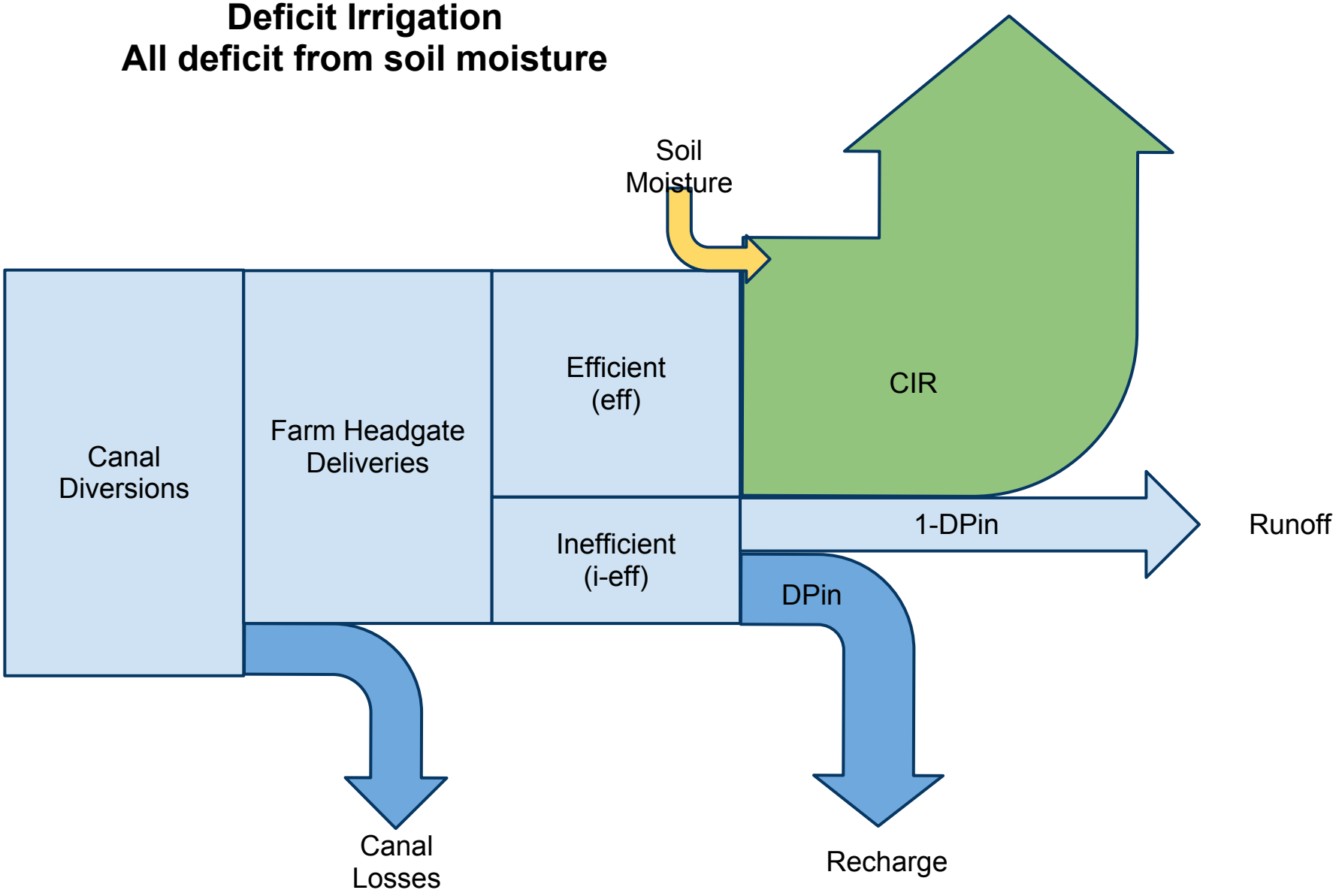
- over the years, the **mkmod** code has been run and tested many times; however, since PERL code is not as readily "readable" by engineers as languages like Visual Basic or FORTRAN, not many people have undertaken the task of reviewing the code directly
- IDWR requested help in reviewing the code & I offered to get permission from Rangen Inc to spend some $ to take a shot at reviewing the PERL code.
- Focused on reviewing code logic (especially farm budget code) and not data testing (yet)

Current Status

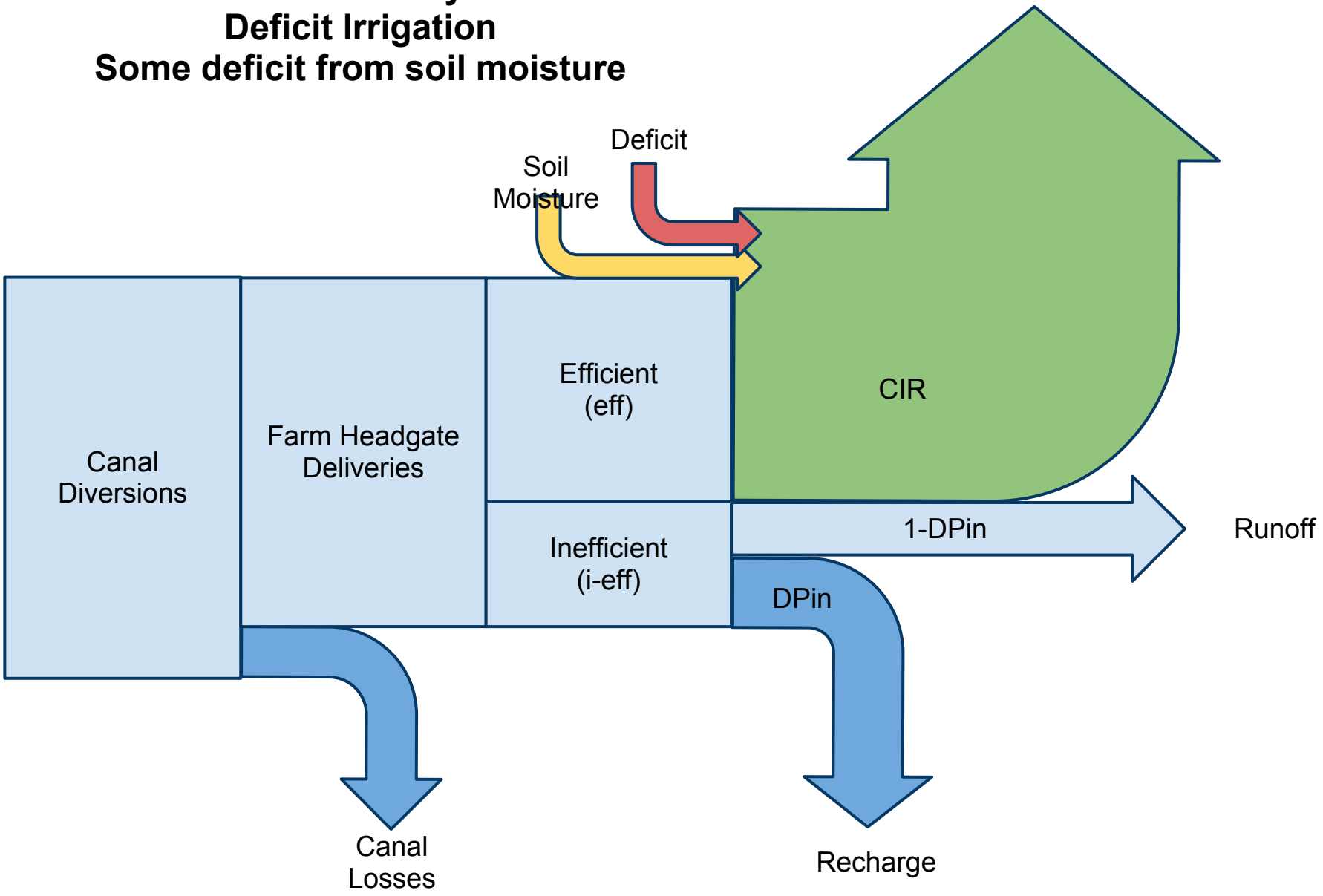
- Gotten much better at reading and understanding PERL code!
- Continued to review the code at a high level to gain a general understanding of the program flow, data structures, and main subroutines.
- Focused on the Irrig Acreage components (Max Effic Farm Budget code, etc.).
- Added comments during my review to a copy of the code
- Prepared diagrams of the farm water budget inferred by the code in under various scenarios
- There is a LOT more to review, and hopefully we'll get more volunteers as IDWR starts the ESPAM 2 training.

Review the code and diagrams

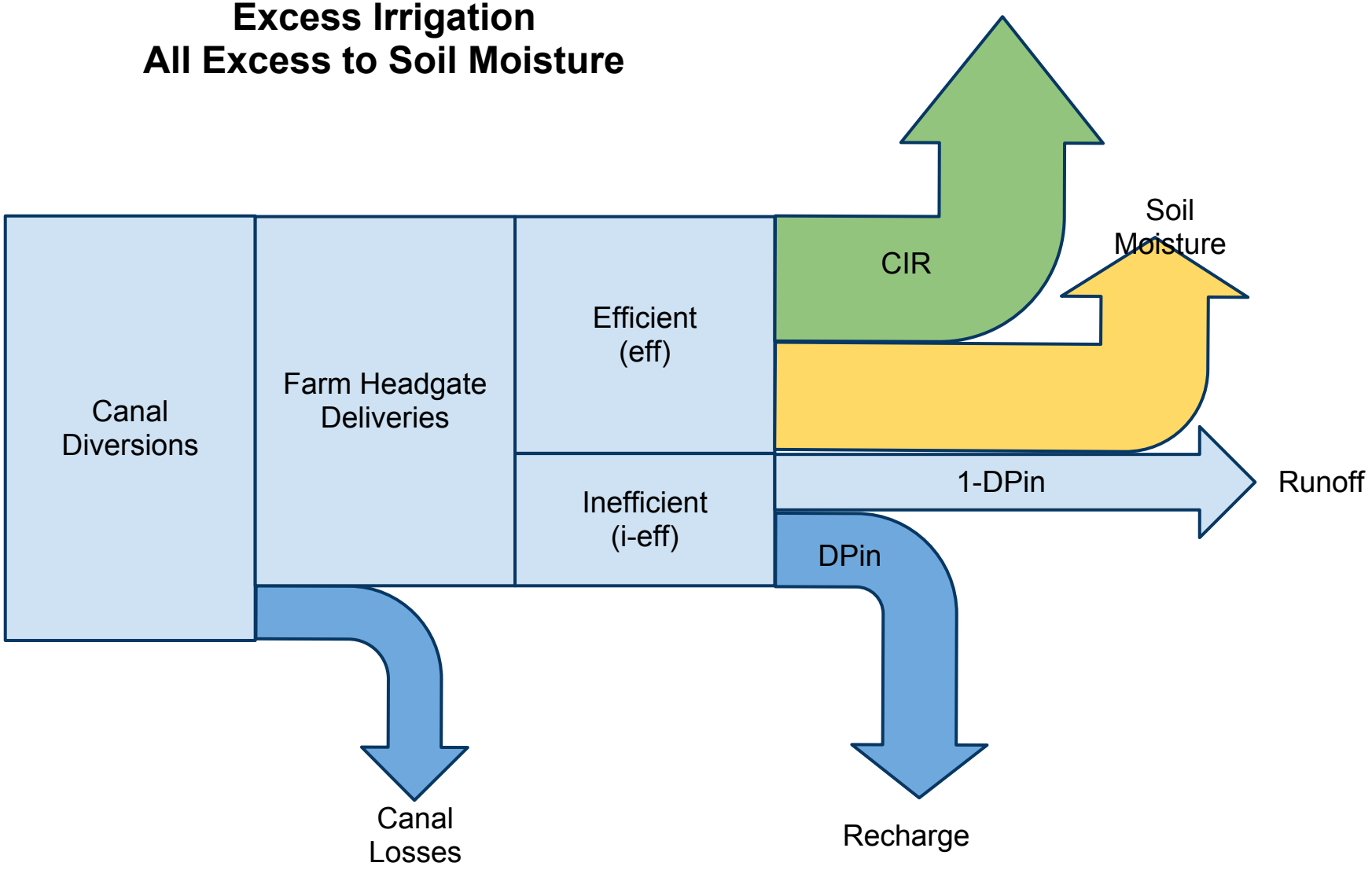
mkmod5 code
Max Farm Efficiency Method
Deficit Irrigation
All deficit from soil moisture



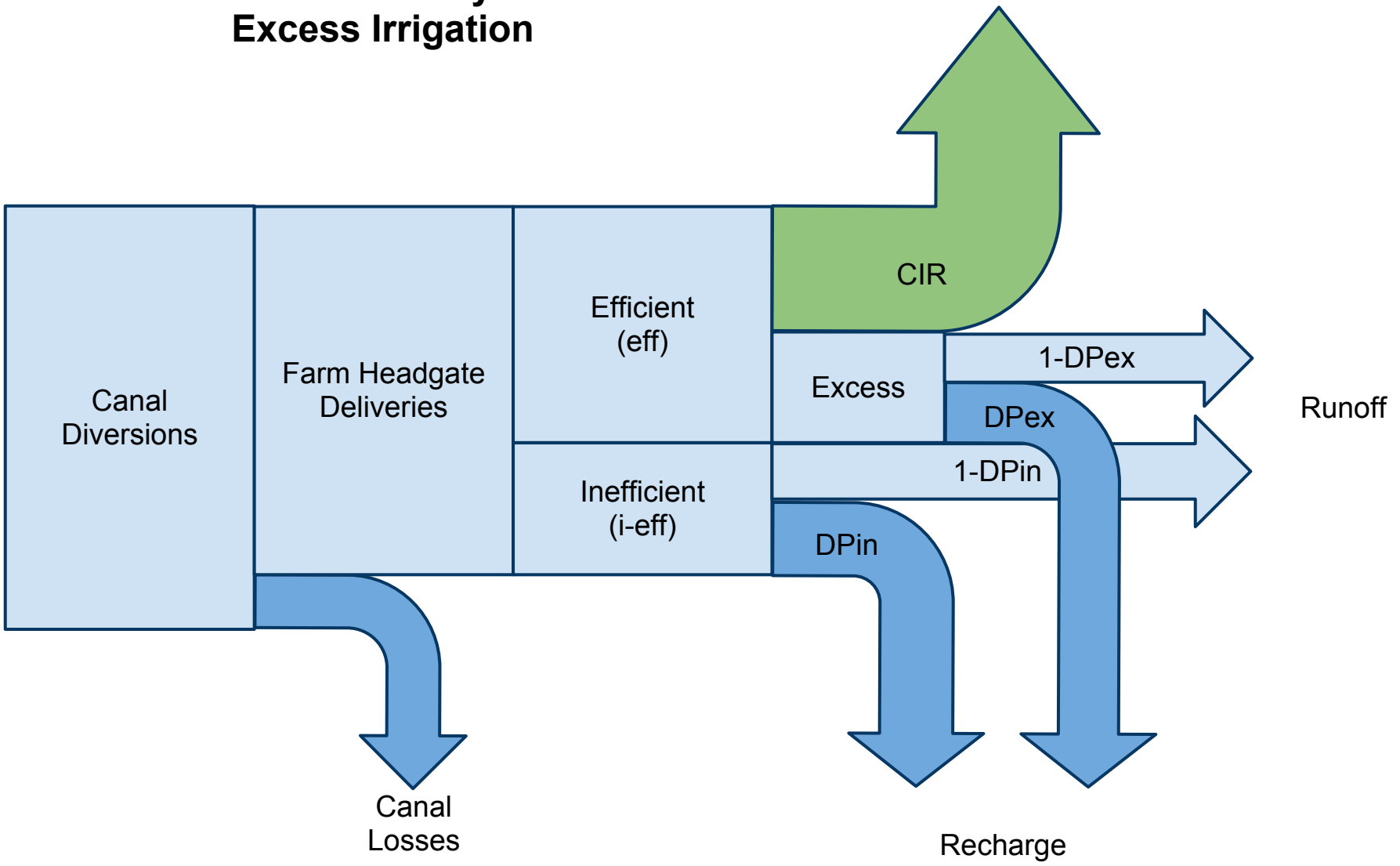
mkmod5 code
Max Farm Efficiency Method
Deficit Irrigation
Some deficit from soil moisture



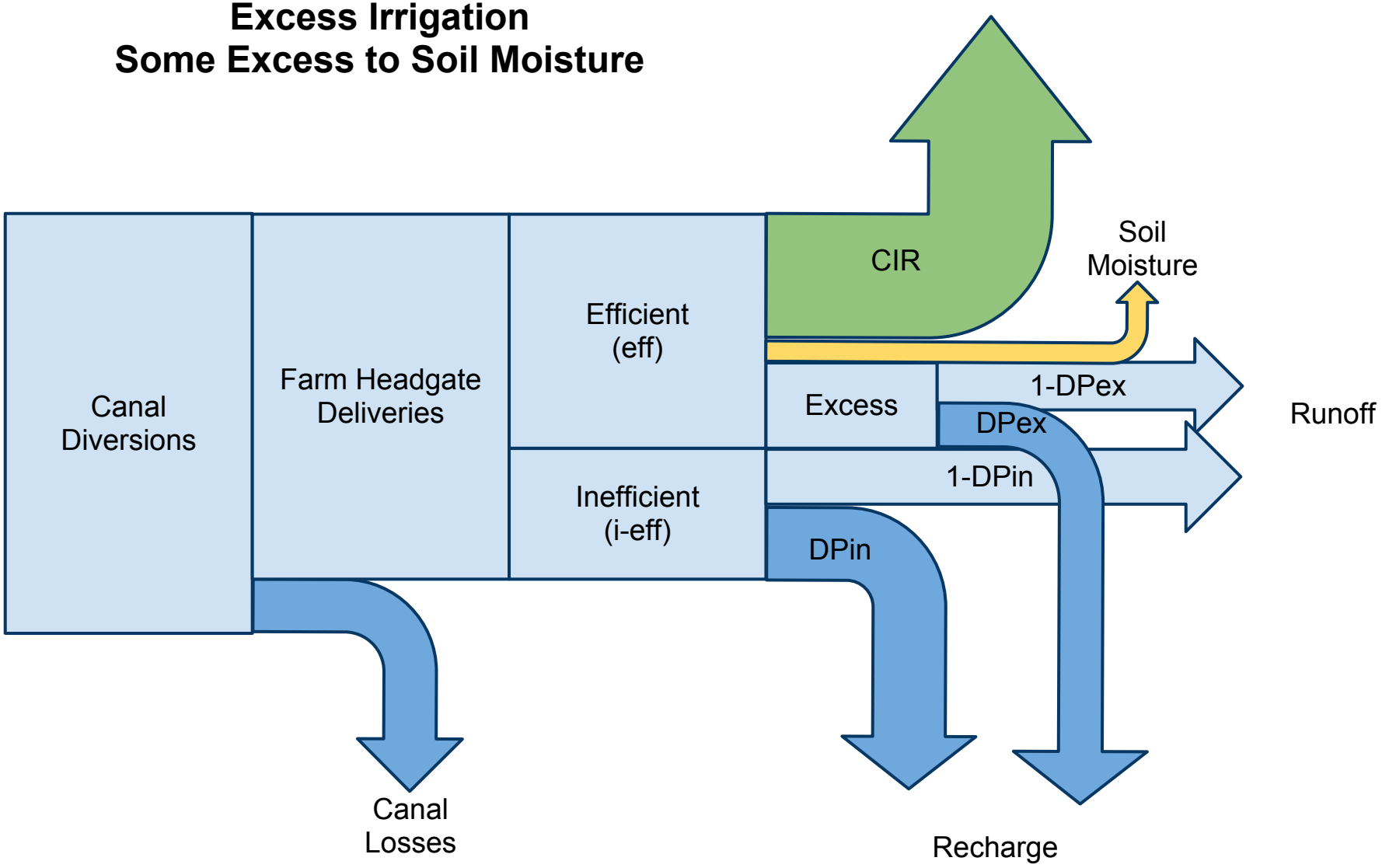
mkmod5 code
Max Farm Efficiency Method
Excess Irrigation
All Excess to Soil Moisture



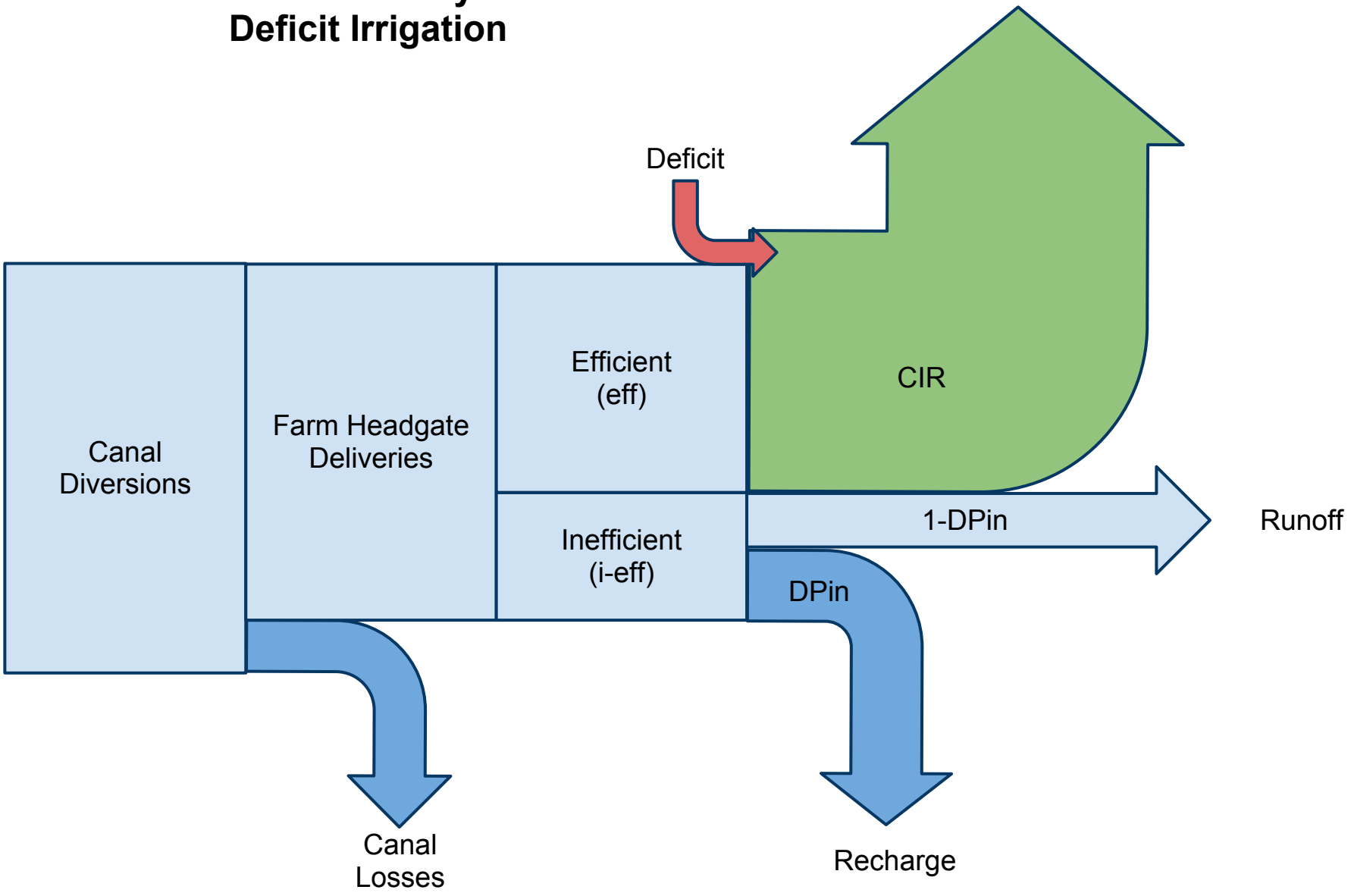
mkmod5 code
Max Farm Efficiency Method
Excess Irrigation



mkmod5 code
Max Farm Efficiency Method
Excess Irrigation
Some Excess to Soil Moisture



mkmod5 code
Max Farm Efficiency Method
Deficit Irrigation



```
 #!/usr/bin/perl -w

#  Create MODFLOW input files for ESPAM
#  Tracks total stresses as well as net
#
#  Willem A. Schreuder
#  V1.0  August 2005
#  V2.0  4-May-2010  Read data from model file
#
#  Input file types
#
#  ENTITIES  (entity parameters)
#    Tag  ENTITIES
#    Header lists entities one per line (name, GW/SW, ET adjustment sprinkler, ET adjustment gravity)
#    For each stress period, one value per entity (Fsp = sprinkler fraction)
#  DIV (diversions)
#    Tag  ENTITIES
#    Header lists entity names all on one line
#    For each stress period, two values each on a new line (Qd = diverted amount, Qr = returns)
#  FPOINT (fixed point diversions)
#    Tag POINT ASSOCIATION
#    Header lists locations one per line (E/M/R/W/O , cell (k,i,j), name)
#      W =  Wetland correction applied to non-irrigated precip recharge
#      R =  Deficit irrigation correction applied to irrigation return flows
#      E =  Exchange pumping
#      M =  Mud Lake pumping
#      O =  Off-site pumping
#    For each stress period, one value per point (Q = volume)
#  POINT (point diversions)
#    Tag POINT ASSOCIATION
#    Header lists locations one per line (cell (k,i,j), name)
#    For each stress period, one value per point (Q = volume)
#  LINE (line recharges:  Canals, Tributary inflows, Perched river cells)
#    Tag LINE
#    Header lists lines as a cell count, scalar and (optional) name, followed by a list of cells one (i,j) per line
#    For eac stress period, one value per line (Q or fraction)
#  ARRAY (cell array values:  Precipitation, ET, non-irrigated recharge)
#    No tag or header
#    For each stress period, one value per cel
#  CELL (Irrigated acres by cell)
#    No tag or header
#    For each stress period, list of cells (potentially - in current data all periods are the same)
#    For each cell in there are three lines
#       (i,j,n) where (i,j) is the cell location, and n is the number of entities
#       List of entity names
#       Area covered by each entity

# jhb notes ######################################################
# the top part of this code was reviewed mainly for general understanding of the data structures
#       and the functions/subroutines that are used in the code following it,
#       it should continue to be checked, but the farm water budget calculations were a higher priority
# jhb notes ######################################################

use strict;
use lib '/pm/grflib/perl';
use GetOpt;
```

```
use MODFLOW;

my $input = 'combo4';
my $output = 'snake20';
my @ext = ('mdl','cel','sol','red','ent','fpt','off','div','cnl','trb','pch','pre','eti','nir','iar','eff');
my %args = GetOpt('i:o:m:sa '.join(' ',map {$_.':'} @ext),i=>$input,o=>$output,m=>'1',s=>0);
(@ARGV>0) && die "Usage: $0\n";
my $group = 1;
my $single = $args{s};

#  Surface water method
my %SWM = (1=>"ESPAM 1.1",MFE=>"Maximum Farm Efficiency using Fixed Returns",FER=>"Maximum Farm Efficiency using Runoff for Returns",FRS=>"Maximum Farm Efficiency using Runoff for Returns and
Soil Moisture");
my $SWM = $args{m};
exists($SWM{$SWM}) || die "Invalid surface water method $SWM\n";
#       logic here is critical to farm calculations later!!!!
#               these only apply to SW entities
#       method 1 is ESPAM 1.1 and all the others use the max farm effic method
my $MaxEff = ($SWM eq 'MFE' || $SWM eq 'FER' || $SWM eq 'FRS');
#       method 1 and MFE have FIXED return flows, the others calculate return flows
#               question - what does the MFE method mean?  where there are "some" fixed return flows in addition to the calculated ones?
my $EffRet = ($SWM eq 'FER' || $SWM eq 'FRS');
#       FRS uses max farm effic, calculated return flows AND is the only method with soil moisture
my $SM     = ($SWM eq 'FRS');

#
#  Set default input file names
#
$output = $args{o};
$input = $args{i};
my %input;
foreach my $ext (@ext)
{
   $input{$ext} = $args{$ext} ? $args{$ext} : "$input.$ext";
}

#  Global variables
my %iar; #  Irrigated Entity Areas
my %x;   #  Input data

# jhb notes ##########################################################
#       define some functions for reading text files
# jhb notes ##########################################################
#
#  Read next non-blank line
#     Parameters:
#       fd  - file handle to read from
#       err - error message to print if line cannot be read
#     Returns:
#       line read or undef if EOF
#       dies on EOF if error message is specified
#
sub Read
{
```

```
   my ($fd,$err) = @_;
   while (my $line = <$fd>)
   {
      chomp $line;
      ($line =~ /^\s*$/) || return $line;
   }
   $err && die "$err\n";
}

#  Read next non-blank line and return parameters split on spaces
#     Parameters:
#       fd - file handle to read from
#     Returns:
#       parameters read or () if EOF
#
sub ReadLine
{
   my ($fd,$err) = @_;
   my $line = Read($fd,$err);
   return split(' ',$line);
}

# jhb notes ##########################################################
#       read the model configuration input file
# jhb notes ##########################################################
#
#  Model configuration
#
open(DAT,"<$input{mdl}") || die "Cannot open file $input{mdl}\n";
#  Read title and units
my $title1 = Read(*DAT,"Error reading title1 from $input{mdl}");
my $title2 = Read(*DAT,"Error reading title2 from $input{mdl}");
my ($Tunit) = ReadLine(*DAT,"Error reading time unit from $input{mdl}");
my ($Lunit) = ReadLine(*DAT,"Error reading length unit from $input{mdl}");
#  Read number of stress periods and steady state definition
my ($Np,$ss0,$ss1) = ReadLine(*DAT,"Error reading number of periods from $input{mdl}");
#  Read stress period lengths and descriptions
my @dt = (0);
my @date = ('Steady State');
my @num = ('--');
for (my $k=1;$k<=$Np;$k++)
{
   $num[$k] = $k;
   ($dt[$k],$date[$k]) = ReadLine(*DAT,"Error reading stress period $k from $input{mdl}");
}
#  Read dimensions
my ($Ni) = ReadLine(*DAT,"Error reading number of rows from $input{mdl}");
my ($Nj) = ReadLine(*DAT,"Error reading number of columns from $input{mdl}");
my ($Nk) = ReadLine(*DAT,"Error reading number of layers from $input{mdl}");
my ($Kp) = ReadLine(*DAT,"Error reading groundwater pumping layer from $input{mdl}");
#  Adjustment factors
my ($Ns) = ReadLine(*DAT,"Error reading number of soils from $input{mdl}");
my @adj = ReadLine(*DAT,"Error reading adjustment factors from $input{mdl}");
#  Wetland adjustment factor
my $WetAdj = pop @adj;
(@adj==$Ns) || die "Incorrect number of adjustment factors in $input{mdl} $Ns\n";
#  Dryland recharge adjustment factor (by soil type)
```

```
my %DryRechAdj;
for (my $k=0;$k<@adj;$k++)
{
  $DryRechAdj{$k+1} = $adj[$k];
}
close(DAT);

#
#  Set up steady state
#
#  Array of periods used for steady state
my @ss = defined($ss0) && defined($ss1) ? ($ss0 .. $ss1) : ();
($single>1 && @ss>0) && warn "<<<WARNING>>>  Steady State omitted\n";
#  Initial period (0 when using SS, 1 if just transient)
my $Np0 = @ss>0 ? 0 : 1;
#  Calculate length of SS period (if there is one)
foreach my $per (@ss)
{
   $dt[0] += $dt[$per];
}

#
#  Set up model domain
#
#  Model Dimensions
my $CellArea = 5280*5280;            #  Cell area (ft^2)
#  Total number of model cells
my $N = $Ni*$Nj*$Nk;
#  MODFLOW object
my $mf = MODFLOW->new(Ni=>$Ni,Nj=>$Nj,Nk=>$Nk);
#  Read IBOUNDs from file
open(DAT,"<$input{cel}") || die "Cannot open file $input{cel}\n";
<DAT> || die "Error reading $input{cel}\n";
my @ibound = ReadCSV(*DAT,"$input{cel}");
close(DAT);
```

jhb notes
a routine for reading comma delimited files
jhb notes

```
#
#  Read array of comma separated data values
#     Reads a value for every grid cell in free format
#     Parameters:
#       fd - file handle to read from
#       which - text string use in error messages
#       ibound - pointer to IBOUND array used to set dead cells to undef
#                if ibound is not specified, data are returned verbatim
#     Returns:
#       array of values in array context
#       pointer to array of values in scalar context
#
sub ReadCSV
{
   my ($fd,$which,$ibound) = @_;
   #  Read values for entire domain
   my @line;
   while (@line<$N)
```

```
   {
      my $line = Read($fd,"Cannot read data from $which");
      push @line , split(',',$line);
   }
   (@line==$N) || die "Incorrect number of data in $which: ".scalar(@line)." instead of $N\n";
   #  Set undef for dead cells if requested
   if ($ibound)
   {
      for (my $k=0;$k<$N;$k++)
      {
         $ibound->[$k] || ($line[$k] = undef);
      }
   }
   return wantarray() ? @line : \@line;
}

# jhb notes ##########################################################
#       a routine for opening and reading misc data files
# jhb notes ##########################################################
#
#  Open file and read header
#     Parameters:
#       file - name of file to open
#       type - type of file (see above)
#       tag  - file starts with tag and entity list, or none if omitted
#       var  - name of variable(s) saved in the file (set when read)
#     Returns:
#       Hash used to access variable(s) in file
#       FD   - file handle to read file
#       FILE - name of file
#       TYPE - type of file
#       VAR  - array of variable names
#       ZERO - variable used to track if data is specified in dead cells (set in Next)
#       DATA - array containing values for stress period (set in Next)
#    This function does if an error is encountered
#
my %ent; #  Global entity pointer
sub Open
{
   my ($file,$type,$tag,@var) = @_;
   my (%data,$Nl);

   # Set up structure
   $data{FD}   = undef;
   $data{FILE} = $file;
   $data{TYPE} = $type;
   $data{VAR}  = \@var;
   $data{ZERO} = 0;
   $data{DATA} = [];

   # An empty file indicates that this type is always all zeroes
   (-e $file && (stat($file))[7]==0) && return \%data;

   # Open file
   my $fd;
   open($fd,"<$file") || die "Cannot open file $file\n";
   $data{FD}   = $fd;
```

```
#  Read tag line and structure count (if requested)
if (defined($tag))
{
   #  Tag line
   my $line = Read($fd,"Cannot read $file");
   ($line =~ /$tag/) || die "$file does not start with $tag\n$line";
   #  Read structure count
   ($Nl) = ReadLine($fd,"Error reading count from $file");
}

#
#  Read the header
#
#  Diversions file lists all structures one line
if ($type eq 'DIV')
{
   my @line = ReadLine($fd,"Premature EOF $type");
   (@line==$Nl) || die "Incorrect nmber of diversion entities\n";
   for (my $l=0;$l<$Nl;$l++)
   {
      $data{DATA}[$l] = {NAME=>$line[$l]};
   }
}
#  Read list of structures - one line per structure
elsif (defined($tag))
{
   #  Read structure
   for (my $l=0;$l<$Nl;$l++)
   {
      my @line = ReadLine($fd,"Premature EOF $type $l");
      #  Read ENTITIES data: name, GW/SW, ET adjustment sprinkler, ET adjustment gravity
      if ($type eq 'ENTITIES')
      {
         my ($nm,$tp,$ETADJsp,$ETADJgr,$text) = @line;
         $data{DATA}[$l] = {TYPE=>$tp,NAME=>$nm,ETADJ=>{SP=>$ETADJsp,GR=>$ETADJgr},TEXT=>$text?$text:''};
      }
      #  Read Fixed POINT data:  type (E/M/R/W/O - see file description) , cell (k,i,j), name
      elsif ($type eq 'FPOINT')
      {
         my ($tp,$k,$i,$j,$nm) = @line;
         my $off = $mf->offset($i,$j,$k);
         !$ibound[$off] && (warn "$file FIXED POINT in dead cell $i $j\n") && $data{ZERO}++;
         $data{DATA}[$l] = {TYPE=>$tp,CELL=>$off,NAME=>$nm};
      }
      #  Read POINT data:  cell (k,i,j), name
      elsif ($type eq 'POINT')
      {
         my ($k,$i,$j,$nm) = @line;
         my $off = $mf->offset($i,$j,$k);
         !$ibound[$off] && (warn "$file POINT in dead cell $i $j\n") && $data{ZERO}++;
         $data{DATA}[$l] = {TYPE=>'O',CELL=>$off,NAME=>$nm};
      }
      #  Read LINE   data: count, scale factor
      #  Read LINENT data: count, scale factor, name
      #    Followed by count lines, each with a cell (i,j)
      elsif ($type eq 'LINE' || $type eq 'LINENT')
      {
         my ($n,$scale,@text) = @line;
```

```
            my $name = ($type eq 'LINENT') ? shift @text : '';
            # Canals must map to valid entity
            ($type eq 'LINENT') && !exists($ent{$name}) && die "Invalid canal entity $name\n";
            my @line;
            for (my $k=0;$k<$n;$k++)
            {
               my ($i,$j) = ReadLine($fd,"Premature EOF LINE $l vertex $k");
               my $off = $mf->offset($i,$j);
               !$ibound[$off] && (warn "$file LINE $name in dead cell $i $j\n") && next;
               push @line , $off;
            }
            $data{DATA}[$l] = {SCALE=>$scale,CELL=>\@line,N=>$n,NAME=>$name,TEXT=>"@text"};
         }
         else
         {
            die "WTF";
         }
      }
   }
   return \%data;
}

# jhb notes ##########################################################
#       a routine that reads the data for the next time step from the files containing
#                 time series data
# jhb notes ##########################################################
#
#  Read next stress period
#     Parameters:
#       data - data structure set by Open used to read file
#       per  - stress period being read
#     The function dies if an error is encountered
#     On exit, the DATA array is populated with values for this stress period
#
sub Next
{
   my ($data,$per) = @_;
   my $fd   = $data->{FD};
   my $file = $data->{FILE};
   my $type = $data->{TYPE};
   my @var  = @{$data->{VAR}};

   #  Short circuit for empty files
   defined($fd) || return;

   # Read header for this stress period
   my $line = Read($fd,"Cannot read STESS PERIOD from $file period $per");
   ($line =~ /STRESS +PERIOD +$per */)  || die "Lost sync in $file period $per\n$line\n";
   # Read indicator (<0 => reuse last period, 0 => all zeroes, >0 => read data)
   my ($ind) = ReadLine($fd,"Cannot read indicator from $file period $per");

   #  Reuse last stress period
   if ($ind<0)
   {
      ($per>1) || die "Cannot reuse data on first stress period in $file\n";
   }
   #  All zeroes
```

```
elsif ($ind==0)
{
   ($type eq 'CELL') && die "ZERO is invalid for CELL data $file\n";
   #  Zero all variables variables for each entity
   if (@var>0)
   {
      foreach my $ent (@{$data->{DATA}})
      {
         @$ent->{@var} = (0) x @var;
      }
   }
   #  Zero array
   else
   {
      @{$data->{DATA}} = (0) x $N;
   }
}
#  Read Array data
elsif ($type eq 'ARRAY')
{
   $data->{DATA} = ReadCSV($fd,"$file period $per",\@ibound);
}
#  Read cell list
elsif ($type eq 'CELL')
{
   #  Erase old data
   $data->{DATA} = [];
   #  Get count
   my ($Nl) = Read($fd,"Cannot read cell count from $file period $per");
   #  Read count entries
   for (my $l=0;$l<$Nl;$l++)
   {
      #  Cell and number of entities
      my ($i,$j,$n) = ReadLine($fd,"Cannot read cell $l from $file period $per");
      my $off = $mf->offset($i,$j);
      #  Entity names
      my @name = ReadLine($fd,"Cannot read entities cell $l from $file period $per");
      (@name==$n) || die "Incorrect entity count cell $l from $file period $per\n";
      #  Areas
      my @area = ReadLine($fd,"Cannot read area cell $l from $file period $per");
      (@area==$n) || die "Incorrect area count cell $l from $file period $per\n";
      #  Map areas to entity names
      my %area;
      for (my $k=0;$k<$n;$k++)
      {
         $area{$name[$k]} += $area[$k];
      }
      #  Ignore dead cells without warning
      $ibound[$off] && push @{$data->{DATA}} , {CELL=>$off,AREA=>\%area};
   }
}
#  Read entity data
else
{
   #  Loop over variables
   foreach my $var (@var)
   {
      #  Read enough values for all entities
      my @line;
```

```
         while (@line<@{$data->{DATA}})
         {
            push @line , ReadLine($fd,"Cannot read $var from $file period $per");
         }
         (@line==@{$data->{DATA}}) || die "Incorrect number of $var in $file period $per: ".scalar(@line)." instead of ".scalar(@{$data->{DATA}})."\n";
         #  Map values to entities
         foreach my $ent (@{$data->{DATA}})
         {
            $ent->{$var} = shift @line;
         }
         #  Zero out POINT data in dead cells if necessary
         if ($data->{ZERO})
         {
            foreach my $ent (@{$data->{DATA}})
            {
               $ibound[$ent->{CELL}] || ($ent->{$var} = 0);
            }
         }
      }
   }
}

#
#  Save to cell format file
#
sub SaveCell
{
   my ($type,$dt,$per,$out) = @_;
   #  Convert volumes to rate
   my $conv = 1/$dt;
   #  Count nonzero entries
   my $n = 0;
   for (my $k=0;$k<$N;$k++)
   {
      $out->[$k] && $n++;
   }
   #  Save data to file
   print DAT "$n STRESS PERIOD $per\n";
   for (my $k=0;$k<$N;$k++)
   {
      $out->[$k] && printf DAT "%d %3d %3d %f\n" , $mf->kij($k) , $conv*$out->[$k];
   }
}

#
#  Save to array format file
#
sub SaveArea
{
   my ($type,$dt,$per,$out) = @_;
   #  Convert volumes to rate per unit area
   #  Convert areas to fraction (0-1)
   my $conv = ($type =~ /^A.W..$/) ? 1/$CellArea : 1/$CellArea/$dt;
   #  Save data to file
   print DAT "1 -1\n";
   print DAT "INTERNAL 1.0 (free) -1 STRESS PERIOD $per\n";
   for (my $k=0;$k<$N;$k++)
   {
      print DAT ' '.$conv*$out->[$k];
```

```
      ($k+1)%$Nj || print DAT "\n";
   }
}
# jhb notes ############################################################
#       read the input data files
# jhb notes ############################################################

#############################################################################
#########  Open data files and read headers and time-invariant values  #########
#############################################################################
#
#  Read soil types
#
open(DAT,"<$input{sol}") || die "Cannot open file $input{sol}\n";
foreach my $k (1,2)
{
   <DAT> || die "Error reading $input{sol}\n";
}
my @soil = ReadCSV(*DAT,"$input{sol}");
#  Map dead cells to zero and check soil type for active cells
my $Na; #  Number of active cells
for (my $k=0;$k<$N;$k++)
{
   #  Dead cell => 0
   if ($ibound[$k]==0)
   {
      $soil[$k] = 0;
   }
   #  Check type
   elsif (!exists($DryRechAdj{$soil[$k]}))
   {
      die "Unknown soil type $soil[$k]\n";
   }
   #  Count active cells
   else
   {
      $Na++;
   }
}
close(DAT);
#
#  Read reduction factors
#
open(DAT,"<$input{red}") || die "Cannot open file $input{red}\n";
#  Gravity
my @REDgr = ReadLine(*DAT,"Cannot read REDgr from $input{red}");
(@REDgr>=$Np) || die "Wrong count REDgr\n";
#  Sprinkler
my @REDsp = ReadLine(*DAT,"Cannot read REDsp from $input{red}");
(@REDsp>=$Np) || die "Wrong count REDsp\n";
close(DAT);
#  Adjust to multiplicative factors with offset 1
@REDgr = (undef,map {1-$_} @REDgr);
@REDsp = (undef,map {1-$_} @REDsp);

#
#  Read irrigation efficiency parameters and soil properties
```

```
#
my (%eff,%DP,%Soil);
open(DAT,"<$input{eff}") || die "Cannot open file $input{eff}\n";
while (my $line = <DAT>)
{
   ($line =~ /^\w*#/) && next;  #  Skip comments
   my ($name,$EffGR,$EffSP,$DPin,$DPex,$wilt,$fc,$depth) = split(' ',$line);
   $eff{$name} = {GR=>$EffGR,SP=>$EffSP,DPin=>$DPin,DPex=>$DPex};
   $Soil{$name} = {wilt=>$wilt,capacity=>$fc,depth=>$depth};
}
close(DAT);

#
#  Read irrigation entities
#
$x{ENT} = Open($input{ent},'ENTITIES','ENTITIES','Fsp');
#  Check data and set up entities pointer
foreach my $ent (@{$x{ENT}{DATA}})
{
   $ent{$ent->{NAME}} = $ent;
   ($ent->{TYPE} eq 'GW' || $ent->{TYPE} eq 'SW')   || die "$ent->{NAME}: Invalid type $ent->{TYPE}\n";
   (0<=$ent->{ETADJ}{SP} && $ent->{ETADJ}{SP}<=1.5) || die "$ent->{NAME}: ETADJ{SP} out of range $ent->{ETADJ}{SP}\n";
   (0<=$ent->{ETADJ}{GR} && $ent->{ETADJ}{GR}<=1.5) || die "$ent->{NAME}: ETADJ{GR} out of range $ent->{ETADJ}{GR}\n";
}
#  Create sorted list of all entity names
my @name = sort keys %ent;

#
#  Read fixed point diversions, off-site pumping and diversions
#
$x{FIX} = Open($input{fpt},'FPOINT','POINT','Q');       #  Fixed point diversions
$x{OFF} = Open($input{off},'POINT','POINT','Q');        #  Off-site groundwater pumping wells
$x{DIV} = Open($input{div},'DIV','ENTITIES','Qd','Qr'); #  Canal diversions and returns
#  Check that off-site wells are assigned to legal surface water entities
foreach my $well (@{$x{OFF}{DATA}})
{
   exists($ent{$well->{NAME}}) || die "Unknown off-site pumping entity $well->{NAME}\n";
   ($ent{$well->{NAME}}->{TYPE} eq 'SW') || die "Off-site pumping entity $well->{NAME} is not SW\n";
}

#
#  Read canals, tributaries and perched streams
#
$x{CNL} = Open($input{cnl},'LINENT','LINE','FRAC');  #  Canals
$x{TRB} = Open($input{trb},'LINE','LINE','Q');     #  Tributary inflows
$x{RIV} = Open($input{pch},'LINE','LINE','Q');     #  Perched river cells

#
#  Open array files
#
$x{PRE} = Open($input{pre},'ARRAY'); #  Precipitation
$x{ETI} = Open($input{eti},'ARRAY'); #  Irrigated ET
$x{NIR} = Open($input{nir},'ARRAY'); #  Recharge on non-irrigated lands
$x{IAR} = Open($input{iar},'CELL');  #  Irrigated acres by cell
```

jhb notes
prior to here - reviewed code for general understanding of data structures used in the following code

```
#       this should continue to be checked, but the following code was a higher priority
# jhb notes ##########################################################

#########################################################################
################  Process data for each transient stress period  ################
#########################################################################

#
#  Loop over transient stress periods
#    Results are saved in @sum and %out so that steady state values can be calculated
#    Everything in this loop is a volume (or equivalently a depth applied to an area)
# jhb notes ##########################################################
#       create the output data structures - the @sum list and %out hash
# jhb notes ##########################################################
my %out; #  Output acres arrays
my @sum; #  Summary data
# jhb notes ##########################################################
#       define the components of the MODFLOW cell net recharge calculations
# jhb notes ##########################################################
my @net = ('WEL','SWR','GWR','PPT','CNL','TRB'); #  Arrays that make up net recharge
#  Open output file for super simple
if ($single>1)
{
   open(DAT , ">$output.net") || die "Cannot open file $output.net\n";
   print DAT "$Na 0 NOPRINT\n";
}
# jhb notes ##########################################################
#       the big time step loop!
# jhb notes ##########################################################
foreach my $per (1 .. $Np)
{
# jhb notes ##########################################################
#       initialize output data structures - here it is the %out hash containing arrays
#               that affect the cell recharge calculations, plus 4 acreage arrays
#               set them to 0
# jhb notes ##########################################################
   #
   #  Initialize output arrays for this stress period
   #
   foreach my $type (@net,'ASWsp','ASWgr','AGWsp','AGWgr')
   {
      @{$out{$type}[$per]} = (0) x $N;
   }

# jhb notes ##########################################################
#       the Next() subroutine is  a major piece of code,
#               with lots of special cases for each kind of input data
#       so far has not been flow charted - just reviewed for general understanding
```

```
#       note that the %x hash is the major input data storage structure
# jhb notes ##########################################################
   #
   #  Read data for this stress period
   #
   foreach my $var (keys %x)
   {
      Next($x{$var},$per);
   }
# jhb notes ##########################################################
#       a little QA/QC - just some value checking
# jhb notes ##########################################################
   #  Check canal seepage rates in [0,1]
   foreach my $cnl (@{$x{CNL}->{DATA}})
   {
      (0<=$cnl->{FRAC} && $cnl->{FRAC}<=1) || die "Canal seepage rate out of range $cnl->{NAME} period $per fraction $cnl->{FRAC}\n";
   }
   #  Check that precip and irrigated ET are non-negative
   #  Non-irrigated recharge may be negative
   foreach my $var ('PRE','ETI')
   {
      for (my $k=0;$k<$N;$k++)
      {
         defined($x{$var}{DATA}[$k]) && ($x{$var}{DATA}[$k]<0)  && warn "Negative $var period $per at $k: $x{$var}{DATA}[$k]\n";
      }
   }

# jhb notes ##########################################################
#       initialize output data structures - here it is the %sum hash containing summaries
# jhb notes ##########################################################
   #
   #  Initialize summaries for this stress period
   #
#       is this a differnet var than the @sum above - or does the code below convert one to the other using contexts?
   my %sum;
   foreach my $name (@name)
   {
      $sum{$name}{Fsp} = $ent{$name}->{Fsp};
      #  Applied, recharge, consumptive use (ET), area
      @{$sum{$name}}{'APP','RCH','ROF','ET','CIR','BCU','AREA'} = (0,0,0,0,0,0,0);
      #  For surface water entities, also diversion, seepage, returns, deficit irrigation and soil moisture
      ($ent{$name}->{TYPE} eq 'SW') && (@{$sum{$name}}{'DIV','SEEP','RET','DEF','EXS','SM'} = (0,0,0,0,0,0));
   }
# jhb notes ##########################################################
#       initialize output data structures - here it is the irrig cell acreages
# jhb notes ##########################################################
   #
   #  Sum total irrigated area by entity
   #  Sum sprinkler and gravity areas by entity by cell
   #  Sum non-irrigated areas by cell
   #
   #  Initialize active cells to cell area, dead cells to zero
```

```
#       cool - uses the ibound (active, inactive) array to init cell irrig acreage
   my @NonArea = map {$_?$CellArea:0} @ibound;
   #  Initialize irrigated area array for each entity to all zeroes
#       sets entity acreages to 0
   foreach my $name (@name)
   {
      @{$iar{$name}} = (0) x $N;
   }
   #  Loop over cells in irrigated area list
   my %IrrArea;
#       here is that %x hash data structure - key to everything
#       loop over the model cells in %x
   foreach my $cell (@{$x{IAR}{DATA}})
   {
      my $off = $cell->{CELL};
      #  Loop over entities in that cell
#       loop through each entity ($name) - acreage pair associated with this cell
      while (my ($name,$area) = each %{$cell->{AREA}})
      {
         #  Adjusted sprinkler and gravity area
#       here is something key to remember - the total irrig acreage for this entity is split into two pieces by the entity's sprinkler fraction
#               i.e. its the same fraction over all cells in the entity - but a different fraction is read in for each time step
         my $Asp =    $sum{$name}{Fsp} *$REDsp[$per]*$area;  # Sprinkler Area =    Sprinkler fraction  * Sprinkler Reduction Factor * Area
         my $Agr = (1-$sum{$name}{Fsp})*$REDgr[$per]*$area;  # Gravity   Area = (1-Sprinkler fraction) * Gravity Reduction Factor   * Area
#       take this away from the cell total acreage (note that doing this over and over will eventually leave nothing but the non irrig acreage left in @NonArea
#               see image
         #  Remove irrigated areas from non-irrigated array
         $NonArea[$off] -= $Asp+$Agr;
#       not necessarily the same number because it could be reduced by reduction factor
         #  Add irrigated area to this entity's irrigated area array
         $iar{$name}[$off] += $Asp+$Agr;
#       keep a total of this entity's total irrig acreage (see prev note)
         #  Add irrigated area to this entity's total area
         $sum{$name}{AREA} += $Agr+$Asp;
#       keep a total of the acreage in each cell irrig by this entity (see prev note)
#       not sure why this has to be +=?  ask willem - are there ever more than one assignment?
         #  Add irrigated area to list of cells irrigated by this entity
         $IrrArea{$name}{SP}{$off} += $Asp;
         $IrrArea{$name}{GR}{$off} += $Agr;
#       same question
         #  Add irrigated area to output array of irrigated area
         $out{'A'.$ent{$name}->{TYPE}.'sp'}[$per][$off] += $Asp;
         $out{'A'.$ent{$name}->{TYPE}.'gr'}[$per][$off] += $Agr;
      }
   }

# jhb notes ##########################################################
#       handles SW deliveries
# jhb notes ##########################################################
   #
```

```
#  Calculate total diversion, applied volume and seepage by surface irrigation entity
#     sum holds totals by entity
#       DIV diverions (including off-site pumping)
#       APP applied water (including off-site pumping, excluding seepage)
#       SEEP canal seepage
#
#  Applied = Diversion - Returns
#     i had to trust for now that the %x hash structure is correctly structured and data-loaded
foreach my $ent (@{$x{DIV}{DATA}})
{
   my $name = $ent->{NAME};
#     important line of code! zeroes out the return flows
   $EffRet && ($ent->{Qr} = 0);  #  Zero Qr when calculating returns
#     could the entity show up multiple times in this loop, and that's why it's a "+=" ?
   $sum{$name}{DIV} += $ent->{Qd};
   $sum{$name}{RET} += $ent->{Qr};
   $sum{$name}{APP} += $ent->{Qd} - $ent->{Qr};
}
#  Add off-site pumping (Q<0)
foreach my $well (@{$x{OFF}{DATA}})
{
   my $name = $well->{NAME};
#     "-=" because its a negative
   $sum{$name}{DIV} -= $well->{Q};
   $sum{$name}{APP} -= $well->{Q};
}
#  Subtract canal seepage
foreach my $cnl (@{$x{CNL}{DATA}})
{
   my $name = $cnl->{NAME};
#     percent of entity SW div served by this canal * canal loss rate * entity total div
#           question - do we ever check that [SCALE]'s sum to 1 in the proprocessing?
   my $seep = $cnl->{SCALE}*$cnl->{FRAC}*$sum{$name}{DIV};
   $cnl->{SEEP} = $seep;
   $sum{$name}{SEEP} += $seep;
   $sum{$name}{APP}  -= $seep;
}

#
#  Calculate application rate for surface water entities
#
foreach my $name (@name)
{
   #  Default value (Groundwater or error)
   $sum{$name}{RATE} = 0;
#     applic rate using APP value is not meaningful on GW entities because it is ONLY SW application
   #  Surface water only (skip others)
   ($ent{$name}->{TYPE} eq 'SW') || next;
   #  Check application rate and irrigated acres
   ($sum{$name}{APP}<0) && (warn "Negative applied diversions $name $sum{$name}{APP} period $per\n") && next;
   ($sum{$name}{APP}>0 && $sum{$name}{AREA}<=0) && ($single || warn "No irrigated area $name period $per\n") && next;
   #  Application rate = Volume / Area
   $sum{$name}{RATE} = ($sum{$name}{AREA}>0) ? $sum{$name}{APP} / $sum{$name}{AREA} : 0;
}
```

```
# jhb notes ##########################################################
#       now put deliv (cal above) and precip onto the entity SW and GW acreages
#               added up from the MODEL CELL data
# jhb notes ##########################################################
   #
   #  Calculate and distribute applied water and precipitation on irrigated lands
   #
   #  Loop over all irrigation entities
#      remember the IrrArea hash should have all the entities in it now
   foreach my $name (sort keys %IrrArea)
   {
      #  Get a pointer to the entity to make the equations simpler
      my $ent = $ent{$name};
      #  Distribute to sprinkler and gravity lands
#      remember these were split out using the GW fraction
      foreach my $type ('SP','GR')
      {
#       there could be multiple pairs of values (cell ID and acreage) for each entity and type
        #  Loop over cells
        while (my ($off,$area) = each %{$IrrArea{$name}{$type}})
        {
           my $rch = 0;       #  Recharge
           my $deficit = 0;   #  CIR deficit
           my $excess  = 0;   #  CIR excess
           my $runoff  = 0;   #  Surface runoff
           my $SMchng  = 0;   #  Change in soil moisture
#       ET is the ENTIRE irrig ET for this CELL (and this time period, right out of the ETI input data) times the adjustment for this entity for this TYPE (GW or SW)
#               assumes this is a depth only over the irrig acreage (not averaged over the whole cell) - question - is this right?
           my $ET  = $ent->{ETADJ}{$type} * $x{ETI}{DATA}[$off];    # Adjusted ET
#       PRECIP is the precip depth for this cell
           my $ppt = $x{PRE}{DATA}[$off];                           # Precip
#       therefore CIR is a depth over only the irrig acreage
#       IMPORTANT TO NOTE: we are using a CELL CIR (depth) and do not consider each entity's acreage separately - in this cell, they all use the same CIR depth
           my $CIR = $ET-$ppt;                                      # Crop Irrigation Requirement
#       assume we now have a correct CIR depth for this cell (and time step)
           #  Groundwater irrigation
           #  Net Recharge = Precip - Adjusted ET = -CIR = Recharge - Pumping
           #  Pumping  = CIR / Irrigation Efficiency
           #  Recharge = Precip + (1-Irrigation Efficiency)*Pumping
#       if this entity is a GW entity, then calculate pumping as a depth, too = CIR/eff and recharge is a depth = pumping-CIR
           if ($ent->{TYPE} eq 'GW')
           {
              my $pmp = $CIR>0 ? $CIR/$eff{$name}{$type} : 0;    # CIR/Eff
              $rch = $pmp-$CIR;                                 # Pumping - CIR
              #
#       ok now - make these depths into volumes by mult by the area for this entity, then add it to the cell's output arrays (%out)
              #  Groundwater recharge for the cell is the rate time area
              $out{GWR}[$per][$off] += $rch*$area;
              #  Groundwater pumping for the cell is the rate time area
              $out{WEL}[$per][$off] -= $pmp*$area;
              #  Add pumping to applied water total for entity
```

```
            $sum{$name}{APP} += $pmp*$area;
         }
#        if this entity is a SW entity, and we are using the Max farm eff method...
         #  Surface water irrigation using Maximum Farm Efficiency method
         #  Net Recharge = Precip + Application - Adjusted ET = Application - CIR
         elsif ($MaxEff)
         {
            #  Calculate excess irrigation or deficit
#        note that previously the application volume (APP) was converted to a RATE - which in this case is equiv to a depth
#           so therefore it is OK to subtract CIR (also a depth) from it
            my $over = $eff{$name}{$type}*$sum{$name}{RATE}-$CIR;
#        note that these are based on the EFFICIENT (effciency applied) HEADGATE (and canal losses already removed) delivery compared to the CIR
            $excess  = ($over>0) ? +$over : 0;
            $deficit = ($over<0) ? -$over : 0;
            #  Adjust for soil moisture
            if ($SM)
            {
               #  Initialize soil moisture to field capacity on first occurrence
               defined($Soil{$name}{$type}{$off}) || ($Soil{$name}{$type}{$off} = $Soil{$name}{capacity});
               #  Soil moisture source
               my $SMsrc = $Soil{$name}{depth}*($Soil{$name}{$type}{$off}-$Soil{$name}{wilt});
               #  Soil moisture sink
               my $SMsink = $Soil{$name}{depth}*($Soil{$name}{capacity}-$Soil{$name}{$type}{$off});
               #  Deficit irrigation may take water from soil moisture
               if ($deficit>0 && $SMsrc>0)
               {
                  $SMchng = ($SMsrc<$deficit) ? -$SMsrc : -$deficit;
                  $deficit += $SMchng;
               }
               #  Irrigation excess may sink water to soil moisture
               elsif ($excess>0 && $SMsink)
               {
                  $SMchng = ($SMsink<$excess) ? $SMsink : $rch;
                  $excess -= $SMchng;
               }
               #  Adjust soil moisture array
               $Soil{$name}{depth} || die "Invalid soil depth $name\n";
               $Soil{$name}{$type}{$off} += $SMchng/$Soil{$name}{depth};
            }
#        the heart of the max effic farm budget - see images based on these equations
            #  Recharge is the sum of initial and excess deep percolation
             $rch = $eff{$name}{DPin}*(1-$eff{$name}{$type})*$sum{$name}{RATE} + $eff{$name}{DPex}*$excess;
            #  Runoff is the sum of initial and excess surface runoff
            $runoff = (1-$eff{$name}{DPin})*(1-$eff{$name}{$type})*$sum{$name}{RATE} + (1-$eff{$name}{DPex})*$excess;
#        convert the depth to a volume, mult by area
            #  Surface water recharge for the cell is the rate time area
            $out{SWR}[$per][$off] += $rch*$area;
         }
#        espam 1.1
         #  Surface water irrigation using ESPAM 1.1 method
         #  Net Recharge = Precip + Application - Adjusted ET = Application - CIR
         else
         {
            $rch = $sum{$name}{RATE} - $CIR;  # Application - CIR
            #  Surface water recharge for the cell is the rate time area
```

```
            $out{SWR}[$per][$off] += $rch*$area;
          }
#       convert all these depth values to volumes for this entity and store them
          #  Accumulate totals for the entity
          $sum{$name}{ET}  += $area*$ET;
          $sum{$name}{CIR} += $area*$CIR;
          $sum{$name}{RCH} += $area*$rch;
          $sum{$name}{ROF} += $area*$runoff;
          $sum{$name}{DEF} += $area*$deficit;
          $sum{$name}{EXS} += $area*$excess;
          $sum{$name}{SM}  += $area*$SMchng;
#       This is the actual CU (incl water to soil moisture) from surface sources as written
#             question - is this what was intended?
          $sum{$name}{BCU} += $area*($CIR-$deficit+($SMchng>0?$SMchng:0));
        }
      }
   }
# jhb notes ##########################################################
#       now calculate the other (non irrig) components
# jhb notes ##########################################################

#       non irrig acreage - uses a precip adj based on soil type (not ET)
   #
   #  Precipitation on non-irrigated lands
   #  Recharge = Soil Type Adjustment * Rate * Area
   #
   for (my $k=0;$k<$N;$k++)
   {
      ($NonArea[$k]<0) && warn "Negative Non-Irrigated Area $per $k $NonArea[$k]\n";
      #  Skip dead cells and cells fully irrigated
      ($soil[$k]==0 || $NonArea[$k]<=0) && next;
      #  Adjust precipitation rate for soil type
      my $ppt = $DryRechAdj{$soil[$k]} * $x{NIR}{DATA}[$k];
      #  Precipitation recharge is rate time non-irrigated area
      $out{PPT}[$per][$k] +=  $ppt * $NonArea[$k];
      #  Accumulate total precipitation recharge and non-irrigated area
      $sum{NONIRR}{APP}  += $ppt*$NonArea[$k];
      $sum{NONIRR}{AREA} += $NonArea[$k];
   }

   #
   #  Fixed point and off-site adjustments
   #
   my %adj = (
     W => 'PPT' ,  #  Wetland correction applied to non-irrigated precip recharge
     R => 'SWR' ,  #  Deficit irrigation correction applied to irrigation return flows
     E => 'WEL' ,  #  Exchange pumping
     M => 'WEL' ,  #  Mud Lake pumping
     O => 'WEL' ,  #  Off-site pumping
     );
   foreach my $grp ('FIX','OFF')
   {
      #  Loop over the locations
      foreach my $loc (@{$x{$grp}{DATA}})
      {
```

```
      #  Array to adjust
      my $type = $loc->{TYPE};
      exists($adj{$type}) || die "Unknown fixed point type $type for $loc->{NAME}\n";
      #  Magnitude of adjustment (WetAdj is a global and is currently 1 so this does nothing)
      my $q = ($type eq 'W' ? $WetAdj : 1) * $loc->{Q};
      #  Accumulate adjustment to output array
      $out{$adj{$type}}[$per][$loc->{CELL}] += $q;
      #  Accumulate magnitude of the adjustment
      $sum{ADJ}{$type} -= $q;
   }
}

#
#  Distribute canal seepage
#  Amount is calculated based on diversion above
#
#  Loop over canals
foreach my $cnl (@{$x{CNL}{DATA}})
{
   #  Distribute uniformly over cells
   my $q = $cnl->{SEEP} / @{$cnl->{CELL}};
   #  Add rate to cells
   foreach my $cell (@{$cnl->{CELL}})
   {
      $out{CNL}[$per][$cell] += $q;
   }
}

#
#  Distribute tributary underflow and perched river recharge
#
foreach my $type ('TRB','RIV')
{
   #  Loop over entities
   foreach my $ent (@{$x{$type}{DATA}})
   {
      #  Distribute uniformly over cells and apply scale factor
      my $q = $ent->{SCALE} * $ent->{Q} / @{$ent->{CELL}};
      #  Add rate to cells
      foreach my $cell (@{$ent->{CELL}})
      {
         $out{TRB}[$per][$cell] += $q;
      }
   }
}

#
#  Calculate application rate for groundwater water entities
#
foreach my $name (@name)
{
   #  Groundwater water only - skip others
   ($ent{$name}->{TYPE} eq 'GW') || next;
   #  Check irrigated acres
   ($sum{$name}{APP}>0 && $sum{$name}{AREA}<=0) && (warn "No irrigated area $name period $per\n") && next;
   #  Application rate = Volume / Area
   $sum{$name}{RATE} = ($sum{$name}{AREA}>0) ? $sum{$name}{APP} / $sum{$name}{AREA} : 0;
}
```

```
#
#  Summarize results by GW/SW
#
foreach my $name (@name)
{
   my $type = $ent{$name}->{TYPE};
   $sum{$type}{APP}  += $sum{$name}{APP};
   $sum{$type}{ET}   += $sum{$name}{ET};
   $sum{$type}{CIR}  += $sum{$name}{CIR};
   $sum{$type}{BCU}  += $sum{$name}{BCU};
   $sum{$type}{RCH}  += $sum{$name}{RCH};
   $sum{$type}{ROF}  += $sum{$name}{ROF};
   $sum{$type}{AREA} += $sum{$name}{AREA};
   #  Surface water only variables
   if ($type eq 'SW')
   {
      $sum{$type}{DIV}  += $sum{$name}{DIV};
      $sum{$type}{RET}  += $sum{$name}{RET};
      $sum{$type}{SEEP} += $sum{$name}{SEEP};
      $sum{$type}{DEF}  += $sum{$name}{DEF};
      $sum{$type}{EXS}  += $sum{$name}{EXS};
      $sum{$type}{SM}   += $sum{$name}{SM};
   }
   #  Sum sprinkler area
   $sum{$type}{Fsp}  += $sum{$name}{Fsp} * $sum{$name}{AREA};
}
#  Compute sprinkler fraction by type
$sum{SW}{Fsp} = $sum{SW}{AREA} ? $sum{SW}{Fsp}/$sum{SW}{AREA} : 0;
$sum{GW}{Fsp} = $sum{GW}{AREA} ? $sum{GW}{Fsp}/$sum{GW}{AREA} : 0;
#  Compute average application rate
foreach my $type ('SW','GW','NONIRR')
{
   $sum{$type}{RATE} = $sum{$type}{AREA} ? $sum{$type}{APP}/$sum{$type}{AREA} : 0;
}

#
#  Calculate irrigation efficiency (CIR/Applied)
#
foreach my $name ('SW','GW',@name)
{
   $sum{$name}{EFF} = ($sum{$name}{APP}>0 && $sum{$name}{BCU}>0) ? $sum{$name}{BCU} / $sum{$name}{APP} : 0;
}

#
#  Calculate net recharge
#  Net = Non-irrigated precip + Surface Recharge + Groundwater Recharge + Canal Leakage + Tributaries - Pumping
#
for (my $k=0;$k<$N;$k++)
{
   foreach my $type (@net)
   {
      $out{NET}[$per][$k] += $out{$type}[$per][$k];
   }
}

#  If single stress no steady state save to file an forget all arrays
if ($single>1)
{
   SaveCell('NET',$dt[$per],$per,$out{NET}[$per]);
```

```
      %out = ();
   }
   #  If single stress out forget the other arrays to save memory
   elsif ($single)
   {
      foreach my $type (keys %out)
      {
         ($type eq 'NET') || ($out{$type}[$per] = undef);
      }
   }

   #  Save summary values for this period
   $sum[$per] = \%sum;
}

################################################################################
#######################  Save results to output files  #########################
################################################################################

#  Select just the net recharge array, or individual components for output
my @out = ($single==1) ? ('NET') : (keys %out);
#  Loop over output files
foreach my $type (@out)
{
   #
   #  Calculate steady state totals for each cell
   #  These are volumes so just add them up
   #
   foreach my $per (@ss)
   {
      for (my $k=0;$k<$N;$k++)
      {
         $out{$type}[0][$k] += $out{$type}[$per][$k];
      }
   }

   my $file = $output.'.'.lc($type);
   open(DAT , ">$file") || die "Cannot open file $file\n";
   if ($type eq 'WEL' || $single && $type eq 'NET')
   {
      #  Maximum is all active cells
      print DAT "$Na 0 NOPRINT\n";
      #  Loop over stress periods (Steady State + Transient)
      foreach my $per ($Np0 .. $Np)
      {
         SaveCell($type,$dt[$per],$per,$out{$type}[$per]);
      }
   }
   #  Array format
   else
   {
      #  Recharge to layer 1
      print DAT "1 0\n";
      #  Loop over stress periods (Steady State + Transient)
      foreach my $per ($Np0 .. $Np)
      {
         SaveArea($type,$dt[$per],$per,$out{$type}[$per]);
      }
   }
```

```
   close(DAT);
}
#  END HERE on -sss (minimal output - useful for PEST runs)
($single>2) && exit;

#
#  Save service areas to file if requested (-a flag)
#
if ($args{a})
{
   foreach my $name (keys %iar)
   {
      open(DAT , ">$name.dat") || die "Cannot open file $name.dat\n";
      print DAT "$name\n";
      for (my $k=0;$k<$N;$k++)
      {
         print DAT ' '.$iar{$name}[$k]/43560;
         ($k+1)%$Nj || print DAT "\n";
      }
   }
}

#
#  Return period
#
sub date
{
   my ($per) = @_;
   return wantarray() ? ($per>$Np?0:$per,$date[$per]) : ($per<1 || $per>$Np ? "<td align=left colspan=2>$date[$per]</td>" : "<td>$num[$per]</td><td align=left>$date[$per]</td>");
}

#
#  Return data for period and entity
#
sub data
{
   my ($name,$per) = @_;
   #  Unit conversions
   my %units = (DIV=>1/43560,SEEP=>1/43560,RET=>1/43560,ROF=>1/43560,APP=>1/43560,ET=>1/43560,CIR=>1/43560,DEF=>1/43560,EXS=>1/43560,SM=>1/43560,RCH=>1/43560,AREA=>1/43560,W=>1/43560,R=>1/
43560,E=>1/43560,M=>1/43560,O=>1/43560,Fsp=>100,RATE=>12,EFF=>100);
   #  Print formats
my %fmt =
(DIV=>'%d',SEEP=>'%d',RET=>'%d',ROF=>'%d',APP=>'%d',ET=>'%d',CIR=>'%d',DEF=>'%d',EXS=>'%d',SM=>'%d',RCH=>'%d',AREA=>'%d',W=>'%d',R=>'%d',E=>'%d',M=>'%d',O=>'%d',Fsp=>'%.1f',RATE=>'%.2f',EFF=>'%.
1f');
   #  Surface water entity (has diversions)
   my @var;
   if (exists($sum[$per]{$name}{DIV}))
   {
      @var = ('DIV','SEEP','RET','ROF','APP','ET','CIR','DEF','EXS','SM','RCH','AREA','Fsp','RATE','EFF');
   }
   #  Groundwater entity (has ET)
   elsif (exists($sum[$per]{$name}{ET}))
   {
      @var = ('APP','ET','CIR','RCH','AREA','Fsp','RATE','EFF');
   }
   #  Non-irrigated areas (has "applied" water)
   elsif (exists($sum[$per]{$name}{APP}))
   {
      @var = ('APP','AREA','RATE');
```

```
   }
   #  Adjustments
   else
   {
      @var = ('W','R','E','M','O');
   }
   #  Return array of values or HTM string
   foreach my $var (@var)
   {
      defined($sum[$per]{$name}{$var}) || warn "Undefined $per $name $var\n";
   }
   return wantarray() ? (map {$units{$_}*$sum[$per]{$name}{$_}} @var)
                      : join('',map {sprintf "<td>$fmt{$_}</td>" , $units{$_}*$sum[$per]{$name}{$_}} @var);
}

#
#  Calculate average rates for summary tables
#  To show this as an annual rate, these need to be weighted
#
sub Average
{
   my ($Per,@per) = @_;
   my $Dt=0;
   foreach my $per (@per)
   {
      $Dt += $dt[$per];
   }
   my $F = 365.25/$Dt;
   foreach my $name (@name,'GW','SW','NONIRR','ADJ')
   {
      foreach my $per (@per)
      {
         my $f = $dt[$per]/$Dt;
         #  Time weighted areas
         my $At = exists($sum[$per]{$name}{AREA}) ? $f*$sum[$per]{$name}{AREA} : 0;
         foreach my $var (keys %{$sum[$per]{$name}})
         {
            if ($var eq 'AREA')
            {
               $sum[$Per]{$name}{AREA} += $At;
            }
            elsif ($var eq 'Fsp')
            {
               $sum[$Per]{$name}{Fsp} += $At * $sum[$per]{$name}{Fsp};
            }
            else
            {
               $sum[$Per]{$name}{$var} += $F*$sum[$per]{$name}{$var};
            }
         }
      }
      #  Compute sprinkler fraction
      exists($sum[$Per]{$name}{Fsp}) && ($sum[$Per]{$name}{Fsp} = $sum[$Per]{$name}{AREA} ? $sum[$Per]{$name}{Fsp}/$sum[$Per]{$name}{AREA} : 0);
      #  Compute average application rate
      exists($sum[$Per]{$name}{RATE}) && ($sum[$Per]{$name}{RATE} = $sum[$Per]{$name}{AREA} ? $sum[$Per]{$name}{APP}/$sum[$Per]{$name}{AREA} : 0);
      #  Calculate irrigation efficiency (CIR/Applied)
      exists($sum[$Per]{$name}{BCU}) && ($sum[$Per]{$name}{EFF} = ($sum[$Per]{$name}{APP}>0 && $sum[$Per]{$name}{BCU}>0) ? $sum[$Per]{$name}{BCU} / $sum[$Per]{$name}{APP} : 0);
   }
}
```

```
#  Steady State
Average(0     , @ss);
Average($Np+1 , 1 .. $Np);
$num[$Np+1] = '--';
$date[$Np+1] = 'Average';

#  Headers for SW, GW and adjustment
my %hdr;
@{$hdr{NONIRR}} = map {"<th>$_</th>"} ('Volume<br>(af)','Acres','Rate<br>(inches)');
@{$hdr{SW}}     = map {"<th>$_</th>"}
('Diverted<br>(af)','Canal<br>Seepage<br>(af)','Returns<br>(af)','Runoff<br>(af)','Applied<br>Water<br>(af)','ET<br>(af)','CIR<br>(af)','Irrigation<br>Deficit<br>(af)','Irrigation<br>Excess<br>(
af)','Soil<br>Moisture<br>(af)','Recharge<br>(af)','Acres','Sprinkler<br>(%)','Application<br>Rate<br>(inches)','Efficiency<br>(%)');
@{$hdr{GW}}     = map {"<th>$_</th>"} ('Total<br>Pumping<br>(af)','ET<br>(af)','CIR<br>(af)','Recharge<br>(af)','Acres','Sprinkler<br>(%)','Application<br>Rate<br>(inches)','Efficiency<br>(%)');
@{$hdr{ADJ}}    = map {"<th>$_</th>"} ('Wetland<br>Correction<br>(af)','Deficit<br>Irrigation<br>(af)','Exchange<br>Pumping<br>(af)','Mud Lake<br>Pumping<br>(af)','Off-site<br>Pumping<br>(af)');

#  Print header and column numbers
my $K=0;
sub Header
{
   my ($n,@hdr) = @_;
   #  Header
   print HTM "<tr>".join('',map {@hdr} (1 .. $n))."</tr>\n";
   #  Column numbers
   print HTM "<tr align=center>";
   for (my $k=0;$k<$n*@hdr;$k++)
   {
      printf HTM "<td>(%d)</td>" , ++$K;
   }
   print HTM "</tr>\n";
}

#
#  Save summary information to HTML tables
#
open(HTM,">$output.htm") || die "Cannot open file $output.htm\n";
print HTM "<html><body>\n";
print HTM "<H1>$input: $SWM{$SWM}</H1>\n";
print HTM "<H3>$title1</H3>\n";
print HTM "<H3>$title2</H3>\n";

#
#  Definition table
#
print HTM "<p><table border>\n";
print HTM "<tr><th colspan=3>Non-Irrigated Land</th></tr>\n";
print HTM "<tr><th>Variable</th><th>Description</th><th>Source</th></tr>\n";
print HTM "<tr><td>Volume</td><td>Volume of non-irrigated recharge</td><td>soil factor * precip * non-irrigated area</td></tr>\n";
print HTM "<tr><td>Acres</td><td>Non-irrigated acres</td><td>Unused .IAR</td></tr>\n";
print HTM "<tr><td>Rate</td><td>Average non-irrigated recharge</td><td>Volume/Area</td></tr>\n";

print HTM "<tr><th colspan=3>Groundwater Irrigated Land</th></tr>\n";
print HTM "<tr><th>Variable</th><th>Description</th><th>Source</th></tr>\n";
print HTM "<tr><td>Total Pumping</td><td>Total pumping</td><td>CIR/Efficiency</td></tr>\n";
print HTM "<tr><td>ET</td><td>Evapotranspiration</td><td>.ETI*ETadj*Area</td></tr>\n";
print HTM "<tr><td>CIR</td><td>Crop Irrigation Requirement</td><td>ET-PPT</td></tr>\n";
print HTM "<tr><td>Recharge</td><td>Deep percolation to groundwater</td><td>Total Pumping - CIR</td></tr>\n";
print HTM "<tr><td>Acres</td><td>Groundwater irrigated acres</td><td>.IAR*reduction factor</td></tr>\n";
print HTM "<tr><td>Sprinkler</td><td>Sprinkler irrigation percentage</td><td>.ENT sprinkler fraction</td></tr>\n";
```

```
print HTM "<tr><td>Application Rate</td><td>Groundwater irrigation application rate</td><td>Total Pumping/Area</td></tr>\n";
print HTM "<tr><td>Efficiency</td><td>Groundwater irrigation efficiency</td><td>CIR/Total Pumping</td></tr>\n";

print HTM "<tr><th colspan=3>Surface Water Irrigated Land</th></tr>\n";
print HTM "<tr><th>Variable</th><th>Description</th><th>Source</th></tr>\n";
print HTM "<tr><td>Diverted</td><td>Total Diversion</td><td>.DIV diversions</td></tr>\n";
print HTM "<tr><td>Canal Seepage</td><td>Canal Seepage</td><td>.CNL fraction * .DIV diversions</td></tr>\n";
if ($EffRet)
{
   print HTM "<tr><td>Returns</td><td>Canal returns</td><td>NA</td></tr>\n";
   print HTM "<tr><td>Runoff</td><td>Surface runoff</td><td>Surface runoff</td></tr>\n";
   print HTM "<tr><td>Applied Water</td><td>Applied surface water irrigation</td><td>.DIV diversions - Canal Seepage</td></tr>\n";
}
elsif ($MaxEff)
{
   print HTM "<tr><td>Returns</td><td>Canal returns</td><td>.DIV returns</td></tr>\n";
   print HTM "<tr><td>Runoff</td><td>Surface runoff</td><td>Surface runoff</td></tr>\n";
   print HTM "<tr><td>Applied Water</td><td>Applied surface water irrigation</td><td>.DIV diversions - Canal Seepage - Returns</td></tr>\n";
}
else
{
   print HTM "<tr><td>Returns</td><td>Canal returns</td><td>.DIV returns</td></tr>\n";
   print HTM "<tr><td>Runoff</td><td>Surface runoff</td><td>NA</td></tr>\n";
   print HTM "<tr><td>Applied Water</td><td>Applied surface water irrigation</td><td>.DIV diversions - Canal Seepage - Returns</td></tr>\n";
}
print HTM "<tr><td>ET</td><td>Evapotranspiration</td><td>.ETI*ETadj*Area</td></tr>\n";
print HTM "<tr><td>CIR</td><td>Crop Irrigation Requirement</td><td>ET-PPT</td></tr>\n";
print HTM "<tr><td>Irrigation Deficit</td><td>Unmet CIR</td><td>".($MaxEff?"CIR - Efficiency * Applied Water":"NA")."</td></tr>\n";
print HTM "<tr><td>Irrigation Excess</td><td>Irrigation more than CIR</td><td>".($MaxEff?"Efficiency * Applied Water - CIR":"NA")."</td></tr>\n";
print HTM "<tr><td>Soil Moisture</td><td>Change in Soil Moisture</td><td>".($SM?"Deficit(-) or Excess (+) Irrigation":"NA")."</td></tr>\n";
print HTM "<tr><td>Recharge</td><td>Deep percolation to groundwater</td><td>".($MaxEff?"DPin*Efficiency*Applied + DPex*Excess":"Applied-CIR")."</td></tr>\n";
print HTM "<tr><td>Acres</td><td>Surface water irrigated acres</td><td>.IAR*reduction factor</td></tr>\n";
print HTM "<tr><td>Efficiency</td><td>Surface water irrigation/system efficiency</td><td>(CIR-deficit+increase in soil moisture)/Applied Water</td></tr>\n";
print HTM "</table>\n";

#
#  Summary table
#
print HTM "<p><table border>\n";
#  Entity names
printf HTM "<tr><th rowspan=3 colspan=2>Period</th>".
               "<th colspan=%d>Non-Irrigated Land</th>".
               "<th colspan=%d>Groundwater Irrigated Land</th>".
               "<th colspan=%d>Surface Water Irrigated Land</th>".
               "<th colspan=%d>Adjustments</th></tr>\n" ,
               scalar(@{$hdr{NONIRR}}) , scalar(@{$hdr{GW}}) , scalar(@{$hdr{SW}}) , scalar(@{$hdr{ADJ}});
#  Header
Header(1,@{$hdr{NONIRR}},@{$hdr{GW}},@{$hdr{SW}},@{$hdr{ADJ}});
#  Data
my @col = ('NONIRR','GW','SW','ADJ');
foreach my $per ($Np0 .. $Np+1)
{
   print HTM "<tr align=right>".date($per).join('',map {scalar(data($_,$per))} @col)."</tr>\n";
}
print HTM "</table>\n";

#
#  SW and GW tables
#
```

```
foreach my $type ('GW','SW')
{
   #  Columns per entity
   my $n = scalar(@{$hdr{$type}});
   #  Get names of all entities of this type
   my @names = grep {$ent{$_}->{TYPE} eq $type} @name;
   push @col , @names;
   #  Summary table
   print HTM "<p><table border>\n";
   #  Entity names
   printf HTM "<tr><th rowspan=3>Entity</th><th colspan=$n>Average Values by $type Entity</th></tr>\n";
   #  Header
   Header(1,@{$hdr{$type}});
   #  Data
   foreach my $name (@names)
   {
      print HTM "<tr align=right><td align=left>$name</td>".data($name,-1)."</tr>\n";
   }
   print HTM "</table>\n";
   #  Generate moderate width tables
   while (@names>0)
   {
      my @name = splice @names , 0 , $group;
      print HTM "<p><table border>\n";
      #  Entity names
      printf HTM "<tr><th rowspan=4 colspan=2>Period</th>".join('',map {"<th colspan=$n>$_ $ent{$_}{TEXT}</th>"} @name)."</tr>\n";
      #  Parameters by entity
      print HTM "<tr>";
      foreach my $name (@name)
      {
         print HTM "<th colspan=$n>ET Adjustment: Sprinkler $ent{$name}->{ETADJ}{SP} Gravity $ent{$name}->{ETADJ}{GR}";
         $MaxEff && print HTM "<br>Maximum Efficiency: Sprinkler $eff{$name}{SP} Gravity $eff{$name}{GR}";
         $MaxEff && print HTM "<br>Deep Percolation Fractions: DPin $eff{$name}{DPin} DPex $eff{$name}{DPex}";
$SM && printf HTM "<br>Field Capacity $Soil{$name}{capacity} Wilting Point $Soil{$name}{wilt} Root Depth $Soil{$name}{depth} feet<br>Soil moisture reservoir %.2f inches" , 12*($Soil{$name}
{capacity}-$Soil{$name}{wilt})*$Soil{$name}{depth};
         print HTM "</th>";
      }
      print HTM "</tr>\n";
      #  Header
      Header(scalar(@name),@{$hdr{$type}});
      #  Data
      foreach my $per ($Np0 .. $Np+1)
      {
         print HTM "<tr align=right>".date($per).join('',map {scalar(data($_,$per))} @name)."</tr>\n";
      }
      print HTM "</table>\n";
   }
}
print HTM "</body></html>\n";
close(HTM);

#
#  Save summary information to table for plotting
#
open(DAT,">$output.dat") || die "Cannot open file $output.dat\n";
foreach my $per ($Np0 .. $Np+1)
{
   printf DAT '%3d' , $per;
   foreach my $val (map {data($_,$per)} @col)
```

```
   {
      printf DAT ' %10.2f' , $val;
   }
   print DAT "\n";
}
close(DAT);

#
#  Save runoff to ASCII table
#
my @SW = grep {$ent{$_}->{TYPE} eq 'SW'} @name;
open(DAT,">$output.rfl") || die "Cannot open file $output.rfl\n";
print DAT "#   Period      ";
foreach my $name (@SW)
{
   printf DAT ' %10s' , $name;
}
print DAT "\n";
foreach my $per ($Np0 .. $Np+1)
{
   printf DAT '%3d %-12s' , date($per);
   foreach my $val (map {$sum[$per]{$_}{ROF}/43560} @SW)
   {
      printf DAT ' %10.2f' , $val;
   }
   print DAT "\n";
}
close(DAT);
```