MetroScope Gen 3.5 Function Definitions

Draft 4/24/2013 Jim Cser, Metro Economic and Land Use Forecasting

G35_IO_functions.R

MetroScope data input/output functions

readTable()

Reads table from CSV format file

Input Parameters:

scenarioID = scenario ID number

relPath = path relative to home directory

tableName = name of CSV file

Calls:

NONE

Returns:

R data frame

writeTable()

Writes array to CSV format file

Input Parameters:

scenarioID = scenario ID number

dataArray = array to be written

relPath = path relative to home directory

tableName = name of CSV file

Calls:

NONE

Returns:

NONE

updateResDemandData_yearN()

Writes residential demand module arrays to temporary directory

Input Parameters:

resDemandResults
   $avgHouseSizeBin = average house size, by housing bin
   $avgLotSizeBin = average lot size, by housing bin
   $avgHedonicBin = average hedonic price, by housing bin

Calls:
writeTable()

Returns:
NONE

updateNonresDemandData_yearN()
Writes non-residential demand module arrays to temporary directory

Input Parameters:
nonresDemandResults
   $sqftDemandEzRe = sqft demand

Calls:
writeTable()

Returns:
NONE

outputResData_yearN()
Writes outputs of residential module

Input Parameters:
resDemandResults
   $res_demand_Rz = DU demand by rzone
   $res_demand_KHIARz = DU demand by rzone, KHIA
   $res_demand_EzRz = DU demand by ezone, rzone
   $res_demand_RzBin = DU demand by ezone, housing bin
   $res_demand_Ez = DU demand by ezone
   $avgHouseSizeBin = average house size, by housing bin
   $avgLotSizeBin = average lot size, by housing bin
   $avgHedonicBin = average hedonic price, by housing bin
   $res_demand_binshares = DU demand shares by housing bin
resSupplyResults
   $newSupply = new regular DU supply increment
   $newSupplyUR = new UR DU supply increment
   $res_supply_Rz = total DU supply

   $acresConsumed = new regular acres consumed
   $acresConsumedUR = new UR acres consumed
   $acresRemaining = total regular acres remaining
   $acresRemainingUR = total UR acres remaining
resLocpriceResults
   $res_locationprice_new = updated location price

Calls:
writeTable()

Returns:
NONE

outputNonresData_yearN()
Writes outputs of residential module

Input Parameters:
nonresDemandResults
   $sqftDemandcalc = total sqft demand
   $empDemand_calc = total employment demand by ezone, emplcass, retype
   $empDemandEzEc = total employment demand by ezone, emplcass
   $empDemandEz = total employment demand by ezone, emplcass
nonresSupplyResults
   $sqftNewSupply = new regular sqft supply increment by ezone, emplcass, retype
   $sqftNewSupply_UR = new UR sqft supply increment by ezone, emplcass, retype
   $sqftNewSupplyEzRe = new regular sqft supply increment by ezone, retype
   $sqftNewSupplyEzRe_UR = new regular sqft supply increment by ezone, retype
   $sqftVintageSupplyEzRe = total sqft supply from previous model year
   $acresConsumed = new regular acres consumed
   $acresConsumedUR = new UR acres consumed
   $acresRemaining = total regular acres remaining
   $acresRemainingUR = total UR acres remaining
nonresLocpriceResults
   $nonres_locationprice_new = updated location price

Calls:
writeTable()

Returns:
NONE

G35_nonres_calcNonresLocationPrice.R

calcNonresLocationPrice()
Uses the calculated sqft supply and demand to adjust the non-residential location price.

Input Parameters:
iLoops = iteration of nonres module
supply_sqft = total nonres sqft supply
demand_sqft = total nonres sqft demand

Calls:
NONE

Returns:
totalSumSq = diagnostic, sum of (demand - supply)^2
nonres_locationprice_new = new nonres location price
nonres_locationprice = old nonres location price

G35_nonres_demand_calcNonresAccess.R

calcNonresAccess()
Calculates non-residential travel access weights

Access is an inverse function of travel time-- the longer the commute, the less probablity of choosing that location

access weights for total employment = total employment [ezone] /
(travel time [ezone1 x ezone2] * time param + travel time ^2 [ezone1 x ezone2] * timesq param

access weights for total households = = base year total households [ezone] /
(travel time [ezone1 x ezone2] * time param + travel time ^2 [ezone1 x ezone2] * timesq param

access weights for employment by employment class = empclass emp [ezone] /
(travel time [ezone1 x ezone2] * time param + travel time ^2 [ezone1 x ezone2] * timesq param

... except for Government- Education access [empclass 15]
where access weights [ezone] = coeff * childhh [ezone] ^ exponent

Input Parameters:
nonres_traveltime_in = nonres travel times
nonres_vintage_emp_in = employment from previous model year
res_currentyear_hh_in = households by ezone
access_param_in = emp, emplcass emp, and hh access parameters
access_time_param_in = travel time access parameters

Calls:
NONE

Returns:
combined_access_weights = total of access weight components

G35_nonres_demand_module.R

runNonresDemand()
The non-residential demand module for MetroScope.

This function distributes the regional control total of employment by employment classes. Both employment demand and square footage demand are calculated and distributed over employment zones and real estate types.

Non-residential demand =
  Employment Demand *
  Baseline Square Feet per Employee (by emp class, re type) *
  Location Price (by ezone, re type) ^ Direct Sqft Elasticity (by emp class, re type) *
  Location Price (by ezone, re type) ^ Direct Price Elasticity (by emp class)

Details of terms of the non-residential demand equations =
  employment controls for year N [empclass] * baseline distribution param [empclass x re] *
  product over re (location price [ez x re] ^ price elasticity [eclass x re x re2]) [empclass x re] *
  baseline sqft/emp [empclass x re] * location price [ez x re] ^ direct sqft elasticity [empclass x re] *
  location price [ez x re] ^ direct price elasticity [empclass] * access weights

access weights =
  (access weight, total emp [ez] * access_param_totalemp [empclass]) +
  (access weight, total hh [ez] * access_param_totalhh [empclass]) +
  (access weight, empclass emp [ez x empclass] * access_param_empclass [empclass])

where [ ] denotes the dimension of the input matrix

Input Parameters:
iLoops = current iteration of nonres demand module

Calls:
calcNonresAccess()

Returns:
sqftDemandcalc = sqft demand by ezone, empclass, retypes
sqftDemandEzRe = sqft demand by ezone, retypes
empDemand_calc = emp demand by ezone, empclass, retypes
empDemandEzEc = emp demand by ezone, empclass
empDemandEz = emp demand by ezone

G35_nonres_supply_calcNonresSupply.R

calcNonresSupply()

This function does the actual calculations for the non-residential supply module .
It gets called twice- once for "regular' supply (non-subsidized) and "UR" supply (subsidized).

Basic steps for calculating non-residential supply:
Calulate the cost to build non-residential real estate for each ezone, far class, and real estate type
Read the nonres demand land and location price from the demand module and prorate it over the real estate ty[es
Then build or no build: if price offered is more than the cost to build, then something gets built; otherwise it doesn't.
Calculate acres consumed by scaling the nonres acres supply by the calculated sqft supply.

Input Parameters:
locationPrice = current nonres location price
ezoneLocationPriceCalib = nonres location price from the calibration base year
cbd = cbd factor, whatever this means
landVal = land value per sqft
farClassParam = parameters related to FAR class
capitalCost = capital cost per sqft
landCost = land cost per sqft
capitalSubsidy = capital subsidy per sqft for UR supply, if applicable
reParams = parameters related to real estate type
nonres_general_params = general parameters for the nonres model
acresStock = stock acres, both regular and UR, rolled over from the previous model year
acresAdded = acres added, both regular and UR, in the current model
initSalesFract = default fraction applied to the current year available acres -- the "throttle"
nonres_demand_yearN = current year sqft demand, calculated by the nonres demand module

Returns:
acresConsumed = acres consumed, by ezone, re type, far
finalSupply = sqft supply, by ezone, re type, far
finalSupplyEzRe = sqft supply, by ezone, re type. Used to calculate nonres location prices
G35_nonres_supply_module.R

runNonresSupply()
The non-residential supply module for MetroScope.

This module loads the required inputs, and then calls the functions to calculate the nonres supply.
All the heavy lifting is done by calcNonresSupply(), once for non-subsidized "regular" supply and once for subsidized "UR" (for urban renweal) supply.

General steps for calculating non-residential supply:
  Calulate the cost to build non-residential real estate for each ezone, far class, and real estate type
  Read the nonres demand land and location price from the demand module and prorate it over the real estate ty[es
  Then build or no build: if price offered is more than the cost to build, then something gets built; otherwise it doesn't.
  Calculate acres consumed by scaling the nonres acres supply by the calculated sqft supply.

Input Parameters:
iLoops = current iteration of nonres module

Calls:
calcNonresSupply()

Returns:
acresConsumed = nonres regular acres consumed
acresRemaining = nonres regular acres remaining
sqftNewSupply = increment in regular sqft supply compared to previous model year
sqftNewSupplyEzRe = increment in regular sqft supply compared to previous model year, by ezone and retype
acresConsumedUR = nonres UR acres consumed
acresRemainingUR = nonres UR acres remaining
sqftNewSupply_UR = increment in UR sqft supply compared to previous model year
sqftNewSupplyEzRe_UR = increment in UR sqft supply compared to previous model year, by ezone and retype
sqftVintageSupplyEzRe = total sqft supply from previous model year

G35_res_calcResLocationPrice.R

calcResLocationPrice()
Uses the calculated DU supply and demand to adjust the non-residential location price.

Input Parameters:
iLoops = iteration of res module
supply_sqft = total DU supply
demand_sqft = total DU demand

Calls:
NONE

Returns:
totalSumSq = diagnostic, sum of (demand - supply)^2
res_locationprice_new = new res location price
res_locationprice = old res location price

G35_res_demand_functions.R

calcHouseLotSize()
Calculates house size or lot size for each KHIA market segment

Both the house size and lot size calculations have the same functional form but with different parameters, so for efficiency a single function is used. The results are used for calculating the hedonic value matrix

Parameters
marketData = matrix of values for each KHIA market segment
params = coefficients for house size, lot size equations

Calls:
NONE

Returns:
result = lot,house size by KHIA segment, zone, housing type

calcHedonic()
Calculates residential hedonic price matrix

The hedonic matrix is the price households are willing to pay for new housing, based on the average lot size, average house size, neighborhood score, and residential location price

params = parameters for hedonic equation
accessindex = relative measure of the proximity of a zone to all other zones
nscore = residential neighborhood score
houseSize == average house size
lotSize = average lot size
res_location_price = current residential location price
res_calibration_price = residential location price from calibration year

Calls:
NONE

Returns:
result = hedonic matrix by KHIA market segment, rzone, housing type

calcResTravelUtility()
Calculates utility based on residential zone-to-zone travel times

Parameters:
locationPrice = current residential location price
res_traveltime = ezone-to-rzone travel time from transport module
vintageSupply = residential supply from previous model year

Calls:
NONE

Returns:
travelWeight = travel time utility, used in type and tenure choice
travelWeightDenom = travel time utility, used in type and tenure choice

**calcTenureChoice(**)
Determines which households choose to be owners or renters

Parameters:
dwellingUnitsEzoneKHIA = DU demand by ezone and KHIA
travelWeight_denom = travel time utility, used in type and tenure choice
travelWeight = travel time utility, used in type and tenure choice
marketData = matrix of values for each KHIA market segment
params = coefficients for tenure choice equation

Calls:
NONE

Returns:
ownUnits = owner choice, by ezone and KHIA
rentUnits = renter choice, by ezone and KHIA

calcTypeChoice ()
Determines which households choose to live single- or multi-family units

Parameters:
ownUnits = DU demand for owners
rentUnits = DU demand for owners
travelWeightEzHt = travel time utility, used in type and tenure choice
travelWeightEzProd = travel time utility, used in type and tenure choice
marketData = matrix of values for each KHIA market segment
params = coefficients for type choice equation

Calls:
NONE

Returns:
ownUnits = owner choice, by ezone and KHIA
rentUnits = renter choice, by ezone and KHIA

calcLocationChoice()
Determines where households choose to live

Parameters:
locationPrice = current residential location price
ezoneRzoneTravelTime = ezone-to-rzone travel time from transport module
ownKhiaBinByEz = owner demand, by ezone and value bin
rentKhiaBinByEz = owner demand, by ezone and value bin
binshares = TODO describe bin shares
typeChoice = demand by housing type [see calcTypeChoice]
kshare = weight for K demand TODO describe Kshare
res_accessindex = relative measure of the proximity of a zone to all other zones
res_nscore = residential neighborhood score
marketData = matrix of values for each KHIA market segment
params = coefficients for type choice equation

Calls:
NONE

Returns:
res_demand_Rz = demand by rzone and housing type, is compared with supply to adjust location price [see calcResLocationPrice]
res_demand_KHIARz = demand by rzone, KHIA and housing type, output only
res_demand_EzRz = demand by rzone, ezone and housing type, output only
res_demand_Ez = demand by ezone, used by non-res module
res_demand_Ezchild = demand by ezone, used by non-res module

calcBinAvg()
Calculates average of input array, by housing bin

Parameters:
arrayData = matrix of values for each KHIA market segment, same as marketData
ownKhiaBin = owner demand by KHIA, bin
rentKhiaBin = renter demand by KHIA, bin

Calls:
NONE

Returns:
arrayDataBin -- demand by housing type, KHIA, bin

calcBinSum()
Calculates sum of input array, by housing bin

Parameters:
arrayData = matrix of values for each KHIA market segment, same as marketData
ownKhiaBin = owner demand by KHIA, bin
rentKhiaBin = renter demand by KHIA, bin

Calls:
NONE

Returns:
arrayDataBin -- demand by housing type, KHIA, bin

G35_res_demand_module.R

runResDemand()
The residential demand module for MetroScope.

This function distributes the regional control total of dwelling unit demand over the residential zones, housing types, tenure, and KHIA categories. The demand is then compared to the results of the supply module, and difference is used to adjust the residential location price.

General sequence of the residential demand calculations is:
1. Zone-to-zone utility based on travel time
2. Distribution over each KHIA market segment
3. Tenure choice (owners vs. renters)
4. Housing type choice (single- vs. multi-family)
5. Location choice
6. Allocation by zone and housing value bin

#Input Parameters:
iLoops = current iteration of res demand module

Calls:
calcHouseLotSize()
calcHouseLotSize()

calcHedonic()
calcResTravelUtility()
calcTenureChoice()
calcTypeChoice()
calcLocationChoice()
calcBinAvg()
calcBinSum()

Returns:
res_demand_Rz -- DU demand, by rzone. requried to update res location price
res_demand_KHIARz -- DU demand, by rzone and KHIA
res_demand_EzRz -- DU demand, by rzone and ezone
res_demand_Ez -- DU demand, by ezone. Required by nonres model
res_demand_EzChild -- Child DU demand, by ezone. Required by nonres model
avgHouseSizeBin -- Average house size, by rzone and bin. Requried by res supply module
avgLotSizeBin -- Average lot size, by rzone and bin. Requried by res supply module
avgHedonicBin -- Average hedonic price, by rzone and bin. Requried by res supply module
res_demand_RzBin -- DU demand, by rzone and housing bin. Required by nonres model
res_demand_binshares -- share of DU demand in each rzone, for each housing bin and type

G35_res_supply_functions.R

calcResFeasibleSupply()
Calculates the DU supply availble to the residential real estate market.

The raw buildable acres available are the acres remaining from the previous model year plus then new acres added in the current model year. This amount is "throttled" by both an assumed
base fraction (not all units will be put on the market every year) and a price factor (as prices go up, more units go on the market). An effective lot size for each zone class and housing type, a function of location price, then converts the buildable acres into potential new DU supply.

Parameters:
res_general_params = general parameters for residential supply module
res_location_price = residential location price for current model year
res_calibration_price = residential location price for calibration year
res_base_salesfraction = fraction of residential acreage introduced to the model, i.e. the "throttle"
res_landsupply_reg = regular land supply input to the model
res_landsupply_ur = regular land supply input to the model
res_zclass_lotsize = minium and maximum lot sizes for each zoning class

Calls:
NONE

Returns:
acresAvail = regular acres available to the market
acresAvailUR = UR acres available to the market
supplyFeasible = potential new regular DU to be built, subject to the market
supplyFeasibleUR = potential new regular DU to be built, subject to the market
lotPriceChange = parameter to adjust building cost as location price changes

calcResAcresConsumed()
Calculates the acres consumed by residential real estate market

For each housing price bin, compare the cost to build a DU to the amount a household is willing to pay.
If construction cost is less than bid price, then build, otherwise don't build.
Finally, calculate the total acres consumed by prorating by the ratio of built and unbuilt DU supply.

The function is called once for regular acres, and once for UR acres.

Parameters:
res_general_params = general residential parameters
lotPrice = base residential lot price
lotprice_chg = parameter to adjust building cost as location price changes
avgLotSizeBin = average lot size, by housing bin
avgHouseSizeBin = average house size, by housing bin
avgPriceRentBin = average hedonic, by housing bin
res_base_bldgcost = base residential building cost
rzoneFeeSubsidy = fee added or subsidy subtracted from building cost
totalCostFraction = factor to adjust total building cost. UNUSED
supplyFeasible = potential new regular DU to be built
acresAvail = acres available to the market

Calls:
NONE

Returns:
newSupply = new supply built in current year, by rzone
acresConsumed = acres consumed in current model year
newSupplyRzZc = new supply built in current year, by rzone and zone class
newSupplyRzBin = new supply built in current model year, by rzone and housing bin
newSupplyRzBinZc = new supply built in current model year, by rzone, housing bin, and zone class
totalCost = total cost of production
G35_res_supply_module.R

runResSupply()
The residential supply module for MetroScope.

The raw buildable acres available are the acres remaining from the previous model year plus then new acres added in the current model year. This amount is "throttled" by both an assumed
base fraction (not all units will be put on the market every year) and a price factor (as prices go up, more units go on the market).

General steps for res supply calculations:
1) Calculate effective lot size for each zone class and housing type,
2) Use lot size and buildable acres to determine eligible new DU supply.

3) For each housing price bin, compare the cost to build a DU to the amount a household is willing to pay.
4) If construction cost is less than bid price, then build, otherwise don't build.
5) Calculate the total acres consumed by prorating by the ratio of built and unbuilt DU supply.

Input Parameters:
iLoops = current iteration of res demand module

Calls:
calcResFeasibleSupply()
calcResAcresConsumed()

Returns:
newSupply = new units of regular DU supply
newSupplyUR = new units of UR DU supply
res_supply_Rz = total DU supply
acresConsumed = res regular acres consumed in current model year
acresRemaining = res regular acres remaining in current model year
acresConsumedUR = res regular acres consumed in current model year
acresRemainingUR = res UR acres remaining in current model year