RMB3.py

RMB.main()

ReadModflowBinaryV3.py

**def main**():

parserArgs is a Namespace

optArgs is a dictionary assigned to parserArgs

parserArgs,argHelp = defs.getArgsFromParser()

optArgs = vars(parserArgs)

quiet provides non-verbose output

**if not optArgs['quiet']:**

optArgs['quiet'] = True

GUI to select Namefile if not provided on command line

**if not optArgs['namefile']:**

optArgs['namefile'] =

ez.fileopenbox(msg=namMsg,title=title,default='\*.nam',

filetypes=ftypes)

**if optArgs['namefile']:**

(discDict)=mf.setDISfile(path, namfile)

else:

**exit**()

GUI interface option selection

**if optArgs['gui']:**

MFgui.guiArgs(optArgs,argHelp)

Reset noArc to True if not able to import arcpy

**if not optArgs['noArc']:**

import arcpy

Create runSpyderString:

**for arg,val in optArgs.items():**

Define workspace areas, depending upon availability of arcpy functions

**if not optArgs['noArc']:**

MFgis.getSpatialA()

arcpy.env.workspace = MFgis.setWorkspc(optArgs['geodb'])

**else**:

print ('Rasters will be created here: ',optArgs['rasFolder'])

Define Clipping Extents if needed

**if optArgs['clipBox']:**

clips = MFgis.modelClips(optArgs['model'])

Process binary Heads file as from Modflow input files:

**if optArgs['heads']:**

ocFilename = mf.FileByInitials(os.path.join(path, namfile), 'OC')

if (optArgs['model'] in ['C4CDC','NPALM','ECSM','LWCSIM']):

ocFilename = mf.FileByInitials(os.path.join(path, namfile), 'OC')

HeadsUnit = mf.getUnitNum(ocFilename\_full,1,0)

Or

HeadsUnit = mf.getUnitNum(ocFilename\_full,1,0)

headsfile = mf.getFileByNum(os.path.join(path,namfile),HeadsUnit)

mf.readBinHead(headsfile,'HEAD',optArgs)

Process SWI Zeta file:

returned results have not been tested

**if optArgs['zeta']:**

swiFilename = mf.FileByInitials(os.path.join(path,namfile), 'SWI2')

zetaUnit = mf.getUnitNum(swiFilename\_full,1,4)

zetafilename = mf.getFileByNum(os.path.join(path,namfile), zetaUnit)

mf.readBinHead(zetafilename,'HEAD',optArgs)

Process binary TDS concentrations

**if optArgs['conc']:**

concfile = mf.getFileByNum(os.path.join(path,namfile), 201)

mf.readBinHead(concfile,'CONC',optArgs)

Process binary UZF CellxCell Budgets

**if optArgs['uzfcbc']:**

uzfFilename = mf.FileByInitials(os.path.join(path,namfile), 'UZF')

uzfUnit = getUnitNum(uzfFilename\_full,1,6)

uzfcbcfilename = getFileByNum(os.path.join(path,namfile), uzfUnit)

mf.readBinCBC(uzfcbcfilename,None,optArgs)

Process binary Cell by cell Budgets:

**if optArgs['cbc']:**

cbcfilename=mf.identBudFile(path,namfile)

ocFilename = mf.FileByInitials(os.path.join(path, namfile), 'OC')

npers = int(discDict['nperiod'])

nlays = int(discDict['layer'])

OCsp=mf.getSP\_OC(ocFilename\_full, npers, nlays)

noterms = mf.readCBCterms(path, namfile,optArgs) #provides OCsp

mf.readBinCBC(cbcfilename,None,optArgs,OCsp)

Process binary CellbyCell as Flow Vectors:

**if optArgs['vector']:**

if 'OCsp' not in locals():

ocFilename = mf.FileByInitials(os.path.join(path, namfile), 'OC')

npers = int(discDict['nperiod'])

nlays = int(discDict['layer'])

OCsp=mf.getSP\_OC(ocFilename\_full, npers, nlays)

noterms = mf.readCBCterms(path, namfile,optArgs) #provides OCsp

cbcfilename=mf.identBudFile(path,namfile)

if optArgs['terms'] != 'RIGHT|FRONT':

optArgs['terms'] = 'RIGHT|FRONT'

mf.readBinCBC(cbcfilename,'VEC',optArgs,OCsp)

Aggregate current rasters as MEAN, MAX, or MIN

**if optArgs['aggregate']:**

if optArgs['layerStr']:

layerList = mf.parseRange(optArgs['layerStr'])

tifFiles = glob.glob(optArgs['rasFolder']+'\\'+pref+typ+wildCardStr)

L = [np.array(MFgis.rasFile2array(rasFile)) for rasFile in tifFiles]

if optArgs['aggregate'] == 'mean':

summaryRas=np.mean(L,axis=0)

elif optArgs['aggregate'] == 'min':

summaryRas=np.min(L,axis=0)

elif optArgs['aggregate'] == 'max':

summaryRas=np.max(L,axis=0)

elif optArgs['aggregate'] == 'sum':

summaryRas=np.sum(L,axis=0)

MFgis.numPy2Ras(summaryRas, rasName, optArgs, discDict)

MFgis.py

module:: Mfgis synopsis:

Modules ReadModflowBinaryV3.py, MfbinaryData.py and MFgui.py use

module RMFB. MFgis.Mfgis as Mfgis to reference

gdal and arcpy specific function definitions

The Functions all support GIS specific processing

Function definitions:

**def getSpatialA():**

# ARCPY: Check out the ArcGIS Spatial Analyst

**def setWorkspc(geodb):**

# ARCPY: Set base paths for ESRI workspace.

**def clearINMEM():**

# ARCPY: Clean up In\_Memory Features, Tables and Rasters

**def getModel\_SR(model):**

# Get EPSG code selected by model name

return(getModel\_SR[model])

**def modelClips(model):**

# Get coordinate extents selected by model name (xmin, ymin, xmx, ymax)

return(modelClips[model])

**def modelOrigins(model):**

# Get origin coordinates by model name (xmin, ymin)

return(modelOrigs[model])

**def modelStrPeriodInterval(model):**

# Model stress periods represent either Daily or Monthly Periods

return(modelStrPeriodInterval[model])

**def CreateGeoTiff(NewFileName, Array, xsize, ysize,xcoord, ycoord,SR):**

# GDAL: Create a georeferenced raster (TIF) from an array of values

return NewFileName

**def pixelOffset2coord(raster, xOffset,yOffset):**

# GDAL: Calculates the coordinate of a pixel center

return coordX, coordY

**def rasFile2array(rasterFile):**

# GDAL: Return an array read from a single, specified band of a raster file

return array

**def raster2array(raster):**

# GDAL: Return an array read from a single, specified band of a raster

return array

**def array2shp**(array,outFeature,rasterFile,arrName="VALUE",espg=2881):

# GDAL: Create and save an ESRI point shapefile from single raster

**def Two\_array2shp**(array1,array2,outFeature,rasterFile,

arrName1="DIR",arrName2="MAG",csizeMultiplier=1,espg=2881):

# GDAL: Create and save an ESRI point shapefile with 2 values from 2 rasters

**def TwoRas2OnePnt(dirRas,magRas,outFeature,optArgs,**

**arrName1="DIR",arrName2="MAG",csizeMultiplier=1):**

# either GDAL or ARCPY:

# Set up raster data as arrays to create shapefiles from 2 rasters

# representing direction and flow magnitude

**def raster\_X\_coeff(file, file2, coeff, band=1):**

# GDAL: multiplies data from rasterfile by coeff and resaves as new raster

**def noPath (file):**

return(os.path.basename(file))

**def numPy2Ras(npArray, rasName, optArgs, discDict,rasMulti=1.0):**

# GDAL or ARCPY:

# Converts NumPy Array to raster

**def clipRaster(InRastername, optArgs):**

# GDAL or ARCPY:

# Clip raster to specified extents from model defaults or shapefile extent

**def equalObs(x, nbin):**

# Calculate equal-frequency bins

**def setColrRampPrec(band,classMeth,nColors,excludeVals,discDict):**

# GDAL: Calculate appropriate precision of first column of colorFile

return (precision,minVal, maxVal,bin\_edges)

**def createColorfile(raster\_file,colorFile,color,classMeth,reverse,excludeVals,discDict):**

# GDAL: Writes colorFile to be read via gdaldem subprocess function call

**def** **RscriptForRamps**():

# Function returns code which may be ran in R to generate RGB values for color ramps

return code

**def colrRampDef():**

# Defines dataframe with RGB values for color rmp themes

return colorRamps

MFgui.py

module::MFgui synopsis:

Modules ReadModflowBinaryV3.py use

module RMFB. MFgui.MFgui as MFgui

To reference function definitions

The Functions invoked when -gui optional argument provided to RMB3.py

Function definitions:

**def guiTorFOptions(justOptions,optArgs,argHelp,prog):**

# GUI choices for Modflow Binary processing options

return(TrueOptions)

**def guiModel(optArgs,argHelp,prog):**

# GUI selection of Model from available choices

return (optArgs['model'])

**def guiArgVals(justArgs,optArgs,argHelp,prog):**

# GUI value definitions of optional arguments not needing path or filenames

return(NoneVals) #returns specific arguments still not assigned or == None.

**def guiMFterms(MFbudTerms,optArgs):**

# GUI choices for Modflow Binary CDC budget term choices using ex.multchoicebox

return(reply) # reply is a list of budget term names

**def guiGeoVals(wkspcArgs,optArgs,argHelp,prog):**

# GUI argument value definitions of arguments needing GIS path or workspaces

return(DefaultVals) #returns arguments still shown with default values

**def guiGetFilename(getFileArgs,optArgs,argHelp,prog):**

# GUI value definitions for arguments needing filenames

return(FileVals)

**def** **guiArgs**(optArgs,argHelp,prog='ReadModflowBinary'):

# Begin processing using GUI interface

# select True/False data options

TrueOptions = guiTorFOptions(justOptions,optArgs,argHelp,prog)

while not TrueOptions and prog=='ReadModflowBinary':

TrueOptions =guiTorFOptions(justOptions,optArgs,argHelp)

# Choose from an existing model definition

if prog=='ReadModflowBinary':

SelectedModel = guiModel(optArgs,argHelp,prog)

# Identify Arguments which are still = None

if len(justArgs.keys())> 0:

NoneVals = guiArgVals(justArgs,optArgs,argHelp,prog)

for arg in NoneVals:

#Pass oneDict as dict with key: == arg value

secondPass =guiArgVals(oneDict,optArgs,argHelp,prog)

MfbinaryData.py

module::MfbinaryData synopsis:

Modules ReadModflowBinaryV3.py and MFgis.py use

module RMFB.MFbinary.MFbinaryData as mf

To reference function definitions

The Functions read and manipulate Modflow input & output files

Function definitions:

**def noPath (file):**

return(os.path.basename(file))

**def binHdr(hdrType):**

return(AllBinHdr[hdrType]) # Dictionary of MF binary types [HEAD, CONC, CDC, etc]

**def checkExec\_env():**

return (cmdL)

**def stopFn():**

running = False

**def makeTerminateBtn():**

running = True

**def parseRange(astr):**

# Parse a string to create list of values from input range

return sorted(result)

**def FileByInitials(sourcefile,pkgInitials):**

# Read Modflow Namefile searching for package initials

return fileName

**def getUnitNum(file, row\_num, item\_num):**

# read Modflow Output Control file

return unitnum

**def getSP\_OC(file, spMax, nlays):**

# read Modflow Output Control file and return list of Stress Periods

return Ocsp

def getFileByNum(sourcefile,fnumber):

# Read Modflow Namefile searching for unit number

return filename

**def getBASdata(file):**

# Read older BAS file for Modflow discretization info rather than the newer DIS file

return discDict # discritization dictionary

**def getBAS6data(file):**

# Read Modflow BAS6 file to populate the HNOFLO in discDict for inactive cell value

return(discDict)

**def getHDRYdata(file):**

# Read Modflow BAS6 file to populate the HDRY in discDict for inactive cells

return(discDict)

**def getBCFdata(file):**

# Read MF BCF file DIS file is not available to populate discretization dictionary: discDict

return(discDict)

**def getDISdata(file):**

# Populate discretization dictionary from Modflow DIS file

return(discDict)

**def setDISfile(path, namfile):**

# Retrieve filename for MF Discretation file searching NAM file for ‘DIS’

disFilename = FileByInitials(path + "\\" + namfile, 'DIS')

if disFilename.strip() == "":

basFilename = FileByInitials(path + "\\" + namfile, 'BAS')

discDict=getBASdata(basFilename\_full)

bcfFilename = FileByInitials(path + "\\" + namfile, 'BCF')

discDict=getBCFdata(bcfFilename\_full)

else:

discDict = getDISdata(disFilename\_full)

bas6File = FileByInitials(path + "\\" + namfile, 'BAS6')

discDict=getBAS6data(basFilename\_full)

GWFFilename = FileByInitials(path + "\\" + namfile, 'LPF')

if GWFFilename.strip() == "":

GWFFilename = FileByInitials(path + "\\" + namfile, 'BCF')

discDict=getHDRYdata(GWFFilename\_full)

return(discDict)

**def modelDisc():**

# Deconstruct the discDict returning a list of values

return nlays,nrows,ncols,npers,celSz1,cellsz2

**def identBudFile(path,namfile):**

# Determine the filename for the CellxCell budget file

if cbcPkgFilename.strip() == "":

cbcPkgFilename = FileByInitials(os.path.join(path,namfile), 'BCF6')

if cbcPkgFilename.strip() == "":

cbcPkgFilename = FileByInitials(os.path.join(path,namfile), 'BCF')

if cbcPkgFilename.strip() == "":

cbcPkgFilename = FileByInitials(os.path.join(path,namfile), 'UPW')

if cbcPkgFilename.strip() == "":

print(" No supported flow Packages (BCF,BCF6,LPF or UPW) found in NAM file")

exit(86)

cbcUnit = getUnitNum(cbcPkgFullName,1,1)

if int(cbcUnit) == 0:

cbcUnit = getUnitNum(cbcPkgFullName,1,2)

cbcFilename = getFileByNum(os.path.join(path,namfile), cbcUnit)

binfilename = os.path.join(path,cbcFilename)

return(binfilename)

**def readBinHead(binfilename,binType,optArgs):**

# Read Modflow 2D Binary file such as the HEADS and CONC

if optArgs['gui']: makeTerminateBtn()

layerRange = optArgs['layerStr']

strPerRange = optArgs['strStr']

nlays,nrows,ncols,npers,celSz1,cellsz2=modelDisc()

ws1 = optArgs['geodb']

Hdr=binHdr(binType)

if layerRange: layerList = parseRange(layerRange)

for SPval in strPerList:

if optArgs['gui']:

root.update()

if not running: root.destroy()

MFhdr = np.fromfile(binfile,Hdr,count=1,sep='')

dataRead = np.fromfile(file=binfile, dtype=np.float32,

count=knt, sep='').reshape(shape)

MFgis.numPy2Ras(dataRead, rastername,optArgs,discDict)

**def readCBCterms(path,namfile,optArgs):**

# Read CellxCell Budget terms to populate GUI selection list named termlist

binfilename=identBudFile(path,namfile)

nlays,nrows,ncols,npers,celSz1,celSz1=modelDisc()

cbcHdr=binHdr('CBC')

binfile=open(binfilename,'rb')

while iper == firstPer:

MFhdr1 = np.fromfile(binfile,cbcHdr,count=1,sep='')

budget = MFhdr1["TEXT"][0].strip().replace(b" ",b"\_").decode("utf-8")

dataRead = np.fromfile(binfile,np.float32,

count=reclen3d).reshape(shp3d)

if iper == firstPer: termlist.append(budget)

return (termlist)

**def magDirFunc(rFaceSlice, fFaceSlice):**

# Calculates magnitude and direction from rightface and front face flow

tmpdirArray = np.arctan2(fFaceSlice,rFaceSlice)\*180 / np.pi

dirArray = np.where(tmpdirArray > 0.0,tmpdirArray,(tmpdirArray+360.0))

magArray = np.power((np.power(fFaceSlice,2)+np.power(rFaceSlice,2)),.5)

return magArray, dirArray

**def readBinCBC(binfilename,rasType,optArgs,OCsp):**

# Reads MF Binary CellxCell Budgets as NumPy arrays for selected TERMS to make rasters

if optArgs['gui']: makeTerminateBtn()

**def doFlowVec(rFaceSlice):**

# local function definiiton: (details below)

# Processes 2 budget Terms: 'FLOW\_RIGHT\_FACE' & 'FLOW\_FRONT\_FACE' to produce points

termset = optArgs['terms']

layerRange = optArgs['layerStr']

strPerRange = optArgs['strStr']

nlays,nrows,ncols,npers,celSz1,celSz1= modelDisc()

intervalTxt = MFgis.modelStrPeriodInterval(optArgs['model'])

for wantedSP in strPerList:

while iper == wantedSP:

MFhdr1 = np.fromfile(binfile,cbcHdr, count=1,sep='')

budget=MFhdr1["TEXT"][0].strip().replace(b" ",b"\_").decode("utf-8")

dataRead=np.fromfile(binfile,np.float32,reclen3d).reshape(shp3d)

if budget == 'FLOW\_RIGHT\_FACE': rFaceSlices=dataRead

if not strPerList or iper in strPerList:

if ilayer+1 in layerList:

if rasType =='VEC' and budget =='FLOW\_FRONT\_FACE':

rFaceSlice = rFaceSlices[ilayer,:,:].reshape(shape)

doFlowVec(rFaceSlice)

elif rasType =='VEC' and budget == 'FLOW\_RIGHT\_FACE':

pass

elif not optArgs['terms'] or optArgs['terms']=='ALL' or \

budget in termset:

MFgis.numPy2Ras(slice, rastername, optArgs, discDict,rasMulti)

MFgis.clipRaster(rastername, optArgs)

return

**def doFlowVec(rFaceSlice):**

# local function definiiton within readBinCBC:

# Processes 2 budget Terms: 'FLOW\_RIGHT\_FACE' & 'FLOW\_FRONT\_FACE' to produce points

if not optArgs['noArc']: outDir = optArgs['geodb']

if optArgs['noArc']: outDir = optArgs['rasFolder']

if budget == 'FLOW\_FRONT\_FACE':

(magArray, dirArray) = magDirFunc(rFaceSlice, fFaceSlice)

MFgis.numPy2Ras(dirArray, rasDir, optArgs, discDict)

MFgis.numPy2Ras(magArray, rasMag, optArgs, discDict)

MFgis.TwoRas2OnePnt(rasDirFile,rasMagFile,arwFeat,optArgs,

"VALUE","Magnitude",1)

if MFgis.modelClips(optArgs['model']) != (0,0,0,0):

MFgis.clipRaster(rasDir, optArgs)

MFgis.clipRaster(rasMag, optArgs)

MFgis.TwoRas2OnePnt(clprasDirFile,clprasMagFile,clparwFeat,optArgs,

"VALUE","Magnitude",1)

if csizeMultiplier > 1:

MFgis.numPy2Ras(fFaceSlice, rasFFMag, optArgs, discDict)

MFgis.numPy2Ras(rFaceSlice, rasRFMag, optArgs, discDict)

if optArgs['noArc']: # GDAL processing

gdal.Warp(rasFFMagXFile+'.tif', rasFFMagFile+'.tif',

outputType=gdal.GDT\_Float32,xRes=cellsize, yRes=cellsize,

resampleAlg = gdal.GRA\_Average)

gdal.Warp(rasRFMagXFile+'.tif', rasRFMagFile+'.tif',

outputType=gdal.GDT\_Float32,xRes=cellsize, yRes=cellsize,

resampleAlg = gdal.GRA\_Average)

fFXras = MFgis.rasFile2array(rasFFMagXFile+'.tif')

rFXras = MFgis.rasFile2array(rasRFMagXFile+'.tif')

(magArray, dirArray) = magDirFunc(rFXras, fFXras)

MFgis.numPy2Ras(dirArray, rasDirX, optArgs, discDict)

MFgis.numPy2Ras(magArray, rasMagX, optArgs, discDict

else: # ARCPY processing

arcpy.Resample\_management(rasFFMag, rasFFMagX, cellsize, "BILINEAR")

arcpy.Resample\_management(rasRFMag, rasRFMagX, cellsize, "BILINEAR")

fFXras = arcpy.RasterToNumPyArray(rasFFMagX,nodata\_to\_value=0)

rFXras = arcpy.RasterToNumPyArray(rasRFMagX,nodata\_to\_value=0)

(magArray, dirArray) = magDirFunc(rFXras, fFXras)

MFgis.numPy2Ras(dirArray, rasDirX, optArgs, discDict)

MFgis.numPy2Ras(magArray, rasMagX, optArgs, discDict)

MFgis.TwoRas2OnePnt(rasDirXFile,rasMagXFile,arwFeatX,optArgs,

"VALUE","Magnitude",csizeMultiplier)

if MFgis.modelClips(optArgs['model']) != (0,0,0,0):

MFgis.clipRaster(rasDir, optArgs)

MFgis.clipRaster(rasMag, optArgs)

MFgis.TwoRas2OnePnt(clprasDirFile,clprasMagFile,clparwFeat,optArgs,

"VALUE","Magnitude",1)

return