

B. SCRIPTS USED TO ASSIST IN METRO MODEL 2 CALIBRATION

distribute_pumping_by_T.py The following script, *distribute_pumping_by_T.py*, was used during model calibration to adjust the pumping rate distribution for wells that penetrate more than one model layer. Normally this calculation is performed automatically by Groundwater Vistas during the creation of the MODFLOW input files. However, during model calibration this calculation needs to be performed by this script.

```
from math import log10

# input files
in1 = open('pumping_allocation.txt','r')
in2 = open('metro_0121_current.par','r')
# output file intermediate - until number of wells is known
out1 = open('distributed_pumping.txt', 'w')

## for debugging
##out2 = open('check_file.dat', 'w')

# create blank dictionaries
# build the transmissivity dictionary
# number of layers penetrated by well varies
#
# key is well name, contents:
# columns:
# 0 total pumping Q
# 1 ROW
# 2 COL
# 3 1st_Layer#
# 4 Thickness of screen in 1st layer
# 5 K_Zone
# 6 2nd_Layer#
# 7 Thickness of screen in 2nd layer (if applicable)
# 8 K_Zone
# 9 3rd_Layer#
# 10 Thickness of screen in 3rd layer (if applicable)
```

```
# 11 K_Zone
# 12 4th_Layer#
# 13 Thickness of screen in 4th layer (if applicable)
# 14 K_Zone
# 15 5th_Layer#
# 16 Thickness of screen in 5th layer (if applicable)
# 17 K_Zone
# 18 6th_Layer#
# 19 Thickness of screen in 6th layer (if applicable)
# 20 K_Zone
# 21 7th_Layer#
# 22 Thickness of screen in 7th layer (if applicable)
# 23 K_Zone

trans_dict = {}

# hydraulic conductivity dictionary - filled as needed from the
# file containing current parameter values
K_dict = {}

# discard the header lines from the input files
line = in1.readline()
line = in2.readline()

# read the transmissivity input file
while True:
    line = in1.readline().split()

    if len(line) == 0:
        break

    trans_dict[line[0]] = []
    for i in range(1, len(line)):
        trans_dict[line[0]].append(line[i])
        # add hydraulic conductivity parameter name to that dictionary
        # if it is not already there
        if line[i][0:2] == 'kx' and not(K_dict.has_key(line[i])):
```

```
        K_dict[line[i]] = []

# read the current K values
while True:
    line = in2.readline().split()

    if len(line) == 0:
        break

    # if hydraulic conductivity zone name in K dictionary, add the current value
    if K_dict.has_key(line[0]):
        K_dict[line[0]].append(float(line[1]))

# sort the keys of the transmissivity dictionary
observations = trans_dict.keys()
observations.sort()

num_wells = 0

for i in observations:
    T = 0.0
    # parse through the rest of entry in the dictionary to calculate the total T
    for j in range(3,len(trans_dict[i])-2,3):
        # print to screen for debugging
        # print i, j
        # calculate each aquifer's contribution to transmissivity and add to total
        T += ( K_dict[trans_dict[i][j+2]][0] * float(trans_dict[i][j+1]) )
        num_wells += 1
    ##      # print out intermediate result for debugging
    ##      print >> out2, i, trans_dict[i][j+2], K_dict[trans_dict[i][j+2]][0], trans_dict[i][j+1], T

    # parse through the rest of entry in the dictionary again to distribute Q
    for j in range(3,len(trans_dict[i]),3):
        # calculate each aquifer's contribution to transmissivity and add to total
        Q_layer = float( trans_dict[i][0] ) * \
            (( K_dict[trans_dict[i][j+2]][0] * float(trans_dict[i][j+1]) ) / T )
        # print out intermediate result for debugging
```

```
print >> out1, trans_dict[i][j], trans_dict[i][1], \
      trans_dict[i][2], Q_layer, i

in1.close()
in2.close()
out1.close()

# debugging file
##out2.close()

in3 = open('distributed_pumping.txt', 'r')
out3 = open('distributed_pumping.wel', 'w')

cbc_unit = '54'

# print header info to wel package input file
print >> out3, '%10s%10s' % (num_wells, cbc_unit)
print >> out3, '%10s' % (num_wells)

# read the unformatted pumping info and write as formatted input for MODFLOW
while True:
    line = in3.readline().split()

    if len(line) == 0:
        break

    print >> out3, '%10s%10s%10s%10.3f%10s' % \
        (line[0], line[1], line[2], float(line[3]), line[4])

in3.close()
out3.close()
```

The following is an example of the input file *pumping_allocation.txt* needed for the script *distribute_pumping_by_T.py*. Because this file is very long, only a small portion is shown here. The full length version of the file used during model calibration can be found on the CD that accompanies this report. The text file should be tab delimited into columns. For wells where additional layers are penetrated the same format is repeated, listing layer number, thickness of that layer, and K zone of that

layer is subsequent columns.

Well Name	Q	ROW	COL	1 st Layer#	Thickness	K Zone	2 nd Layer#	Thickness	K Zone
18246	4052.4234	186	183	3	47.3416	kx256	4	24.2394	kx342
18247	3775.4944	186	182	3	50.8303	kx256	4	22.6241	kx342
18248	4821.5219	186	182	6	51.581	kx504	7	19.1018	kx601
18296	-1.8726	188	184	3	58.6838	kx256	4	33.9294	kx342
18297	-467.7118	188	184	3	58.6838	kx256	4	33.9294	kx342
18339	-136.3002	135	186	3	52.3757	kx236	4	23.8192	kx331
18356	-447.8997	161	157	3	37.4729	kx201	4	30.9708	kx324
18366	-168.9952	65	84	9	32.5908	kx800			
18377	-165.2838	62	78	8	7.4021	kx707	9	30.3947	kx800
18391	-87.556	179	114	4	22.6992	kx305			
18392	-168.2567	133	180	3	22.5673	kx233	4	20.7288	kx330
18397	-14.0486	167	193	4	30.2823	kx334			
18419	-251.1383	182	143	4	28.3285	kx339			
18421	-4.0519	168	160	3	18.3893	kx225	4	28.3022	kx325
18431	-40.1433	147	173	3	30.903	kx201	4	30.1741	kx324
18435	-1.4972	104	138	6	43.82	kx525	7	15.4451	kx613
18441	-174.9845	136	180	4	23.9172	kx307	5	10.4496	kx420
18457	-34.9236	76	192	6	40.7695	kx520			
18462	-762.1151	215	166	3	38.3352	kx257	4	28.2597	kx343
18464	2068.2585	205	161	4	30.8337	kx343			
18465	-524.7292	146	173	3	30.7365	kx201	4	30.2568	kx324
18473	-778.11	103	163	6	35.1941	kx518			
18474	-874.8855	102	165	4	6.4144	kx302	5	17.5491	kx401

anisotropy_regularisation.py The following script, *anisotropy_regularisation.py*, was used to control the anisotropy ratio during model calibration. Vertical hydraulic conductivity was allowed to change during model calibration. This script was used to assign a residual for the anisotropy ratio for each hydraulic conductivity zone. If the vertical hydraulic conductivity was greater than the horizontal hydraulic conductivity a large residual was assigned. If the vertical hydraulic conductivity was less than the horizontal hydraulic conductivity a small residual was assigned, a large residual was assigned when the vertical hydraulic conductivity was greater than the horizontal hydraulic conductivity. Essentially there was a penalty for vertical hydraulic conductivity being smaller than horizontal hydraulic conductivity.

```
from math import log10, exp

'''
script to calculate modeled vertical anisotropy and impose
penalty for aniso > 1.0

add following template and instruction file references to PEST control file:
metro_0121_current.par.tpl metro_1211_1_current.par

anisotropy_ratios.out.ins    anisotropy_ratios.out

djd
'''

# currently have to hard-wire names of the files
in1 = open('metro_0121_current.par','r')
#in1 = open('junk.par','r')
out1 = open('anisotropy_ratios.out', 'w')

# parameter dictionary
# key is parameter name
# associated list has:
# parameter value as index 0

param_dict = {}

# discard the header line
```

```
line = in1.readline().split()

while True:
    line = in1.readline().split()

    if len(line) == 0:
        break

    param_dict[line[0]] = [ float(line[1]), float(line[2]), float(line[3]) ]

# ? need to sort keys? for sake of making instruction file?
keys = param_dict.keys()
keys.sort()

# implement a scaling scheme similar to Doherty's dry cell correction
# transmissivity calculation

K = 1.0          # saturated hydraulic conductivity - nominal value
t_r = 0.1        # residual thickness - (becomes residual for PEST)
f_scale = 0.95   # scaling factor on f, must be < 1/t_r
f = f_scale / t_r # conductivity decay factor
g = (1 / t_r) - f # 2nd exponential decay coefficient

for i in keys:
    #print >> out1, i[0:1]
    if i[0:2] == 'kx':
        #identify corresponding key for Kz
        #print i
        vert = 'kz'+i[2:]
        #print vert
        #aniso - Kz/Kx - if greater than 1, scale as a residual
        aniso = param_dict[vert][0]/param_dict[i][0]
        t = log10( aniso )
        if t > 0.0:
            resid = K * ( (t_r * exp(-g * t)) + t )
        else:
            resid = K * t_r * exp(f * t)
```

```
print >> out1, 'aniso_'+i[2:], resid, aniso, i  
in1.close()  
out1.close()
```

transmissivity_calculation.py The following script, *transmissivity_calculation.py*, was used to during model calibration to calculate the transmissivity at the location of a pumping test. A residual was then calculated for the difference between model calculated transmissivity to that calculated from the pumping test.

```
from math import log10, exp
...
keywords/phrases:
add to dictionary

script to calculate transmissivity values for single and multiple aquifer
wells

djd
...

# input files
in1 = open('transmissivity_input.dat','r')
in2 = open('metro_0121_current.par','r')
# output file
out1 = open('transmissivity_output.dat', 'w')

# create blank dictionaries
# build the transmissivity dictionary
# key is observation name, contents:
# columns:
# 0 - name of first aquifer K zone
# 1 - thickness of aquifer at point of transmissivity estimate
# 2 - name of 2nd aquifer penetrated by well (if applicable)
# 3 - thickness of 2nd aquifer penetrated by well (if applicable)
trans_dict = {}
# hydraulic conductivity dictionary - filled as needed from the
# file containing current parameter values
K_dict = {}
```

```
# discard the header lines from the input files
line = in1.readline()
line = in2.readline()

# read the transmissivity input file
while True:
    line = in1.readline().split()

    if len(line) == 0:
        break

    trans_dict[line[0]] = []
    for i in range(1, len(line)):
        trans_dict[line[0]].append(line[i])
        # add hydraulic conductivity parameter name to that dictionary
        if line[i][0:2] == 'kx':
            K_dict[line[i]] = []

# read the current K values
while True:
    line = in2.readline().split()

    if len(line) == 0:
        break

    # if hydraulic conductivity zone name in K dictionary, add the current value
    if K_dict.has_key(line[0]):
        K_dict[line[0]].append(float(line[1]))

# sort the keys of the transmissivity dictionary
observations = trans_dict.keys()
observations.sort()

for i in observations:
    T = 0.0
    for j in range(0, len(trans_dict[i]), 2):
        # calculate each aquifer's contribution to transmissivity and add to total
```

```

T += ( K_dict[trans_dict[i][j]][0] * float(trans_dict[i][j+1]) )
# print out intermediate result for debugging
#print >> out1, i+'_int', T
# print out final result for PEST to read
print >> out1, i, T

in1.close()
in2.close()
out1.close()

```

The following is an example of the input file *transmissivity_input.dat* used in the script *transmissivity_calculation.py*. All columns should be tab delimited. The zone column corresponds to the hydraulic conductivity zone that the pumping test is located in. The thickness is the layer thickness at the location of the test for each hydrostratigraphic unit that the pumping test pertains to.

Observation	Zone	Thick1	Zone2	Thick2	Zone3	Thick3
T2349	kx24	15.772				
T1051	kx800	42.03				
T2265	kx76	20.082				
T2066	kx813	39.189				
T2306	kx516	54.019	kx612	14.916		
T2277	kx64	11.805	kx64	2.29		
T2347	kx64	33.446				
T2031	kx524	42.642	kx613	11.672		
T1043	kx518	35.206	kx611	14.888	kx726	42.86
T2336	kx63	19.304				
T1006	kx817	47.842				
T2063	kx64	3.233	kx64	9.762		
T1038	kx229	35.469				
T2039	kx328	29.693				
T2227	kx226	28.529	kx326	25.766		