# The General Equilibrium of Elastic Layered Systems (GELS)
# An Open-Source Implementation in Python

David Yang

May 9, 2024

## 1 Code

**bessel.py**

```
""" MIT License
Copyright 2023 David Yang
Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:
The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
"""

from math import cos, sqrt

"""
Abramowitz and Stegun, Handbook of Mathematical Functions

"""
# Zeros of J1(x), J0(x), J0(zs[2]==2.4048256) == 0 and J1(zs[3]==3.831706) == 0
# If index, i for zs is even, then zs[i] is a J0(zs[i]) zero
# if index, i for zs is odd, then zs[i] is a J1(zs[i]) zero
zs = [
       0.0,      1.0,      2.4048256,3.831706, 5.5200781,7.0155867,8.6537279
    , 10.173468,11.791534,13.323692,14.930918,16.470630,18.071064,19.615859
    , 21.211637,22.760084,24.352472,25.903672,27.493479,29.046829,30.634606
    , 32.18968, 33.775820,35.332308,36.917098,38.474766,40.058426,41.617094
    , 43.199792,44.759319,46.341188,47.901461,49.48261, 51.043535,52.624052
    , 54.185554,55.765551,57.327525,58.906984,60.469458,62.048469,63.611357
    , 65.189965,66.753227,68.331469,69.895072,71.472982,73.036895,74.614501
    , 76.1787,  77.756026,79.320487,80.897556,82.46226, 84.039091,85.604019
    , 87.18063, 88.745767,90.322173,91.887504,93.463719,95.029232,96.605268
    , 98.170951,99.74682,101.31266,102.88837,104.45437,106.02993,107.59606
    ,109.17149,110.73775] # J0(x), J1(x) zeros
# Polynomial coefficients for J0(x) and J1(x) approximations
a07 = [ 1.0,        -2.2499997,  1.2656208, -0.3163866,  0.0444479,  -0.0039444,   0.00021]
a14 = [ 0.79788456, -0.00000077,-0.0055274, -0.00009512, 0.00137237, -0.00072805,  0.00014476]
```

```
a21 = [-0.78539816, -0.04166397,-0.00003954, 0.00262573,-0.00054125, -0.00029333,  0.00013558]
a28 =  [0.5,         -0.56249985, 0.21093573,-0.03954289, 0.00443319, -0.00031761,  0.00001109]
a35 =  [0.79788456,  0.00000156, 0.01659667, 0.00017105,-0.00249511,  0.00113653, -0.00020033]
a42 = [-2.35619449,  0.12499612, 0.0000565, -0.00637879, 0.00074348,  0.00079824, -0.00029166]
def polyx(a, x): # Horner's rule polynomial evaluaton
    y = a[-1]
    for i, ai in enumerate(a[-2::-1]):
        y = y * x + ai
    return y
def bes1(x):
    if -3.0 <= x <= 3.0: # J1(x), Polynomial Approximation, P.370, 9.4.4
        x3 = x / 3.0
        return polyx(a28, x3 * x3) * x
    elif 3.0 < x: # J1(x), Polynomial Approximation, P.370, 9.4.6
        x3 = 3.0 / x
        return polyx(a35, x3) / sqrt(x) * cos(x + polyx(a42, x3))
    else:
        return 0.0
def besz(x):
    if -3.0 <= x <= 3.0: # J0(x), Polynomial Approximation, P.369, 9.4.1
        x3 = x / 3.0
        return polyx(a07, x3 * x3)
    elif 3.0 < x: # J0(x), Polynomial Approximation, P.369, 9.4.3
        x3 = 3.0 / x
        return polyx(a14, x3) / sqrt(x) * cos(x + polyx(a21, x3))
    else:
        return 0.0
def integrate_bessel(f, params):
    # set function, # of values, radius, radial, depth, thickness, modulus, poisson, layer, depths
    nss, ar, rr, zz, hs, es, mus, n, hns = params
    # 4 point Gaussian Integration at xg points with wg weights
    xg, wg = [-0.86113631, -0.33998104, 0.33998104, 0.86113631], [0.34785485,  0.65214515, 0.65214515, 0.34785485]
    # initialize ax to zs[0] == 0.0 and sum to 0.0 for 4 integrals with nss values
    ax, sum = zs[0], [[0.0 for j in range(4)] for i in range(nss)]
    if ar < rr:
        # J1(zs[i    ] * ar / rr) * J0(zs[i    ]), p * ar == zs[i    ] * ar / rr
        # J1(zs[i + 1] * ar / rr) * J0(zs[i + 1]), p * ar == zs[i + 1] * ar / rr
        # J1(zs[i    ] * ar / rr) != 0, J0(zs[i    ]) == 0
        # J1(zs[i + 1] * ar / rr) != 0, J0(zs[i + 1]) != 0
        # last integration when J0(zs[i]) == 0, J1(zs[i + 1]) == 0
        lastzs = zs[1:] # end on a J1 zero
    else:
        # J1(zs[i    ]) * J0(zs[i    ] * rr / ar), p * ar == zs[i    ]
        # J1(zs[i + 1]) * J0(zs[i + 1] * rr / ar), p * ar == zs[i + 1]
        # J1(zs[i    ]) == 0, J0(zs[i    ]) != 0
```

```
    # J1(zs[i + 1]) != 0, J0(zs[i + 1]) == 0
    # last integration when J1(zs[i]) == 0, J0(zs[i + 1]) == 0
    lastzs = zs[1:-1] # end on a J0 zero
for izs, azs in enumerate(lastzs):
    # set bx to next zs
    bx = azs
    if ar < rr:
        # set bx when ar < rr
        bx = bx * ar / rr
    dx = (bx - ax) / 2.0
    ex = dx + ax
    for ixg, axg in enumerate(xg):
        pa = dx * axg + ex
        p = pa / ar
        pr = 0.0
        if 0.0 < rr:
            pr = p * rr
        cx = dx * wg[ixg]
        j1 = bes1(pa)
        j2 = 1.0
        j3 = 0.5
        if 0.0 < rr:
            j2 = besz(pr)
            j3 = bes1(pr)/pr
        fx = f(p, params) # stress/displacement at Hankel transform value
        for i in range(len(fx)):
            fxi = fx[i] * cx * j1
            sum[i][0] += fxi * j2
            sum[i][1] += fxi * j3
            sum[i][2] += fxi * j2 / p
            sum[i][3] += fxi * j3 * p
    # set next ax to bx
    ax = bx
return sum
```

**layer.py**

```
""" MIT License
Copyright 2023 David Yang
Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:
The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
"""

from math import exp

"""
Crawford, Hopkins, Smith, Theoretical Relationships between Moduli for Soil Layers beneath Concrete Pavements

"""
# python lists are index zero based, the equation solution in reference is index one based
# functions are used to convert between reference one based indexes and python zero based indexes
def fp():
    return fp.p
def hn(n): # depth of layer
    return hn.hns[n - 1]
def th(n): # layer thickness
    return th.ths[n - 1]
def ev(n): # layer modulus of elasticity
    return ev.evs[n - 1]
def mu(n): # layer poisson's ratio
    return mu.mus[n - 1]
def emu(n): # layer constant
    return (1.0 + mu(n)) / ev(n)
# layer constants, pages 36 - 38
def beta1(n):
    return emu(n + 1) / emu(n)
def beta2(n):
```

```
    return beta1(n) - 4.0 * mu(n) + 3.0
def beta3(n):
    return 2.0 * fp() * hn(n) - 4.0 * mu(n + 1) + 1.0
def beta4(n):
    return 2.0 * fp() * hn(n) + 4.0 * mu(n + 1) - 1.0
def beta5(n):
    return -2.0 * fp() * hn(n) + 4.0 * mu(n) - 1.0
def beta6(n):
    return beta1(n) * (3.0 - 4.0 * mu(n + 1)) + 1.0
def beta12(n):
    return beta6(n) - 4.0 * (1.0 - mu(n))
def beta7(n):
    return 2.0 * fp() * hn(n) + 4 * mu(n) - 1
def beta8(n):
    return (beta4(n) * beta2(n) - beta7(n) * beta6(n)) / 2.0
def beta9(n):
    return (beta3(n) * beta2(n) + beta5(n) * beta6(n)) / 2.0
def beta10(n):
    return (beta12(n) - beta7(n) * (1.0 - beta1(n)) * beta3(n)) / 2.0
def beta11(n):
    return (-beta12(n) - beta5(n) * (1.0 - beta1(n)) * beta4(n)) / 2.0
def put_betas(N): # create list of betas
    f_betas, betas = [beta1, beta2, beta3, beta4, beta5, beta6, beta7, beta8, beta9, beta10, beta11, beta12], []
    if N > 1:
        for n in range(1, N):
            betan = []
            for f_beta in f_betas:
                betan.append(f_beta(n))
            betas.append(betan)
    return betas
def beta(n, i):
    return beta.betas[n - 1][i - 1]
# return A constants, with A1 = A5 * exp(-2*p*Hn) and A3 = A6 * exp(-2*p*Hn), page 39
def a(n, i):
    if i == 1:
        return a.a_s[n - 1][5 - 1] * exp(-2.0 * fp() * hn(n))
    if i == 3:
        return a.a_s[n - 1][6 - 1] * exp(-2.0 * fp() * hn(n))
    return a.a_s[n - 1][i - 1]
def put_a5(n): # create A5, pages 39 - 40
    sum = beta(n, 2) * exp(-2.0 * fp() * th(n + 1)) * a(n + 1, 5)
    sum += beta(n, 7) * (beta(n, 1) - 1.0) * a(n + 1, 2)
    sum += beta(n, 8) * exp(-2.0 * fp() * th(n + 1)) * a(n + 1, 6)
    sum += beta(n, 10) * a(n + 1, 4)
    sum /= 4.0 * (1.0 - mu(n))
```

```
    return sum
def put_a2(n): # create A2
    sum = beta(n, 5) * (beta(n, 1) - 1.0) * exp(-2.0 * fp() * th(n + 1)) * a(n + 1, 5)
    sum += beta(n, 2) * a(n + 1, 2)
    sum += beta(n, 11) * exp(-2.0 * fp() * th(n + 1)) * a(n + 1, 6)
    sum += beta(n, 9) * a(n + 1, 4)
    sum /= 4.0 * (1.0 - mu(n))
    return sum
def put_a6(n): # create A6
    sum = 2.0 * (1.0 - beta(n, 1)) * a(n + 1, 2)
    sum += beta(n, 6) * exp(-2.0 * fp() * th(n + 1)) * a(n + 1, 6)
    sum += beta(n, 3) * (1.0 - beta(n, 1)) * a(n + 1, 4)
    sum /= 4.0 * (1.0 - mu(n))
    return sum
def put_a4(n): # create A5
    sum = 2.0 * (beta(n, 1) - 1.0) * exp(-2.0 * fp() * th(n + 1)) * a(n + 1, 5)
    sum += beta(n, 4) * (beta(n, 1) - 1.0) * exp(-2.0 * fp() * th(n + 1)) * a(n + 1, 6)
    sum += beta(n, 6) * a(n + 1, 4)
    sum /= 4.0 * (1.0 - mu(n))
    return sum
# set value of ani into A, need to create new list to avoid python reusing previous list
def put_ani(ani, n, i):
    a_s_new = []
    for k, r in enumerate(a.a_s):
        row = []
        for j, c in enumerate(r):
            col = c
            if k == n - 1:
                if j == i - 1:
                    col = ani
            row.append(col)
        a_s_new.append(row)
    return a_s_new
def get_an(n):
    return [a(n, 1), a(n, 2), a(n, 3), a(n, 4), a(n, 5), a(n, 6)]
def get_ani(n, i):
    return a(n, i)
# set A values of layers
def put_a_s(n):
    # indexes and create functions
    a_i  , f_i = [5, 2, 6, 4], [put_a5, put_a2, put_a6, put_a4]
    for i, f in enumerate(f_i):
        a.a_s = put_ani(f(n), n, a_i[i]) # set layer A5, A2, A6, A4
    a.a_s = put_ani(0, n, 1) # set layer A1 to zero
    a.a_s = put_ani(0, n, 3) # set layer A3 to zero
```

```
# Solve for A2 and A4 of bottom layer, page 41
def get_a_s():
    an = [0.0, 1.0, 0.0, 0.0, 0.0, 0.0] # zero based index, A1 to A6
    a.a_s = [[0.0 for i in range(6)] for n in range(len(th.ths))]
    for i, ani in enumerate(an):
        a.a_s = put_ani(ani, len(th.ths), i + 1) # set bottom layer, A
    if len(th.ths) > 1:
        for n in list(range(1, len(th.ths)))[::-1]: # reverse order
            put_a_s(n) # layers > 1
    a21, a22, a23, a24 = get_ani(1, 5), get_ani(1, 2), get_ani(1, 6), get_ani(1, 4) # top layer, A

    an = [0.0, 0.0, 0.0, 1.0, 0.0, 0.0] # zero based index, A1 to A6
    a.a_s = [[0.0 for i in range(6)] for n in range(len(th.ths))]
    for i, ani in enumerate(an):
        a.a_s = put_ani(ani, len(th.ths), i + 1) # set bottom layer, A
    if len(th.ths) > 1:
        for n in list(range(1, len(th.ths)))[::-1]: # reverse order
            put_a_s(n) # layers > 1
    a41, a42, a43, a44 = get_ani(1, 5), get_ani(1, 2), get_ani(1, 6), get_ani(1, 4) # top layer, A

    # page 41, 2 x 2 matrix inversion, solved explicitly for A2 and A4 of bottom layer
    c2, c3 =  exp(-2.0 * fp() * hn(1)), (a21 * a44 - a24 * a41 + a22 * a43 - a23 * a42) * (4.0 * mu(1) - 1.0)
    c4 = (a23 * a44 - a24 * a43) * 4.0 * mu(1) * (2.0 * mu(1) - 1.0)
    c1 = ((a23 * a41 - a21 * a43) * c2 + (a22 * a41 - a21 * a42) * 2.0 + c3 + c4) * c2 + a24 * a42 - a22 * a44
    an2 =  ((a41 + (2.0 * mu(1)) * a43) * c2 + a42 - (2.0 * mu(1)) * a44) / c1
    an4 = -((a21 + (2.0 * mu(1)) * a23) * c2 + a22 - (2.0 * mu(1)) * a24) / c1

    an = [0.0, an2, 0.0, an4, 0.0, 0.0] # set bottom layer, A
    a.a_s = [[0.0 for i in range(6)] for n in range(len(th.ths))]
    for i, ani in enumerate(an):
        a.a_s = put_ani(ani, len(th.ths), i + 1)  # set bottom layer, A
    if len(th.ths) > 1:
        for n in list(range(1, len(th.ths)))[::-1]: # reverse order
            put_a_s(n) # layers > 1
def f_s_all(p, params): # calulate stresses and displacements for Hankel transform variable, p
    nss, ar, rr, zz, hs, es, mus, n, hns = params
    fp.p, th.ths, ev.evs, mu.mus, hn.hns = p, hs, es, mus, hns
    u, c, z = mu(n), emu(n), zz
    eph, epz = exp(-2 * fp() * (hn(n) - z)), exp(-fp() * z)
    if len(hs) > 1:
        beta.betas = put_betas(len(hs))
    get_a_s()
    a1, a2, a3, a4, a5, a6 = get_an(n)
    a5 *= eph
    a6 *= eph
```

```
# Hankel transformed stresses and displacements
szz =  a2 + a4*(fp()*z - 2*u + 1) - a5 - a6*(fp()*z + 2*u - 1)
srz =  a2 + a4*(fp()*z - 2*u) + a5 + a6*(fp()*z + 2*u)
wz  =  c*(-a2 - a4*(fp()*z - 4*u + 2) - a5 - a6*(fp()*z + 4*u - 2))
ur  =  c*(-a2 - a4*(fp()*z - 1) + a5 + a6*(fp()*z + 1))
srr =  -a2 - a4*(fp()*z - 2*u - 1) + a5 + a6*(fp()*z + 2*u + 1)
stt =  2*u*a4 + 2*u*a6
sss =  -a2 - a4*(fp()*z - 1) + a5 + a6*(fp()*z + 1)
return [sa * epz for sa in [szz, srz, wz, ur, srr, stt, sss]]
```

**daygels.py**

```python
""" MIT License
Copyright 2023 David Yang
Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:
The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
"""

from math import log, pi, sqrt
from bessel import integrate_bessel
from layer import f_s_all

def put_hn(hs): # H(n)
    sum, hns = 0.0, []
    for ih in range(len(hs)):
        sum += hs[ih]
        hns.append(sum)
    return hns
def poisson(e):
    # empirical relationship between modulus of elasticity and poisson's ratio
    return 0.65 - 0.08 * log(e) / log(10.0)
def get_result(params):
    ij0, psi, rr = params
    return [-ij0[0][0] * psi, -ij0[1][3] * psi * rr, -ij0[2][2] * psi, -ij0[3][1] * psi * rr,
            -(ij0[4][0] - ij0[6][1]) * psi, -(ij0[5][0] + ij0[6][1]) * psi]
def get_rr(params):
    xwheel, ywheel, xout, yout = params
    xrs, yrs, rrs = [], [], []
    for i, x in enumerate(xwheel):
        xr, yr = xout - xwheel[i], yout - ywheel[i]
        rr = sqrt(xr * xr + yr * yr)
        if rr > 0.0:
```

```
            xr, yr = xr / rr, yr / rr
        else:
            xr, yr = 1.0, 0.0
        xrs.append(xr)
        yrs.append(yr)
        rrs.append(rr)
    return [xrs, yrs, rrs]
def get_radius(params):
    opwgt, wgt, psi = params
    return sqrt(opwgt * wgt / psi / pi)
def run_gels(params):
    vehicle, wheels, output, pavement = params
    opwgt, wgt, psi = vehicle[0], vehicle[1], vehicle[2]
    ar = get_radius((opwgt, wgt, psi))
    xwheel, ywheel, xout, yout = wheels[0], wheels[1], output[0], output[1]
    th, ev, ps = pavement[0], pavement[1], pavement[2]
    hns, rrs, ss = put_hn(th), get_rr((xwheel, ywheel, xout, yout)), []
    results = [[0.0 for j in range(7)] for i in range((len(th) - 1) * 2 + 1)]
    for ir in range(len(xwheel)):
        xr, yr, rr, lz, iz, z1, z2 = rrs[0][ir], rrs[1][ir], rrs[2][ir], [], 0, [], []
        if len(th) > 1:
            for ln, zz in enumerate(hns[:-1]):
                # top of layer
                lz.append(ln + 1)
                z1.append(zz - th[ln])
                z2.append(zz)
                ij0 = integrate_bessel(f_s_all, (7, ar, rr, zz - th[ln], th, ev, ps, ln + 1, hns))
                ssh = get_result((ij0, psi, rr))
                # polar to cartesian
                sxx = ssh[4] * xr * xr + ssh[5] * yr * yr
                sxy = ssh[4] * xr * yr - ssh[5] * xr * yr
                syy = ssh[4] * yr * yr + ssh[5] * xr * xr
                szz, srz, wz, ur = ssh[0], ssh[1], ssh[2], ssh[3]
                for i, s in enumerate([sxx, sxy, syy, szz, srz, wz, ur]):
                    results[iz][i] += s
                iz += 1
                # bottom of layer
                lz.append(ln + 1)
                z1.append(zz)
                z2.append(zz)
                ij0 = integrate_bessel(f_s_all, (7, ar, rr, zz, th, ev, ps, ln + 1, hns))
                ssh = get_result((ij0, psi, rr))
                sxx = ssh[4] * xr * xr + ssh[5] * yr * yr
                sxy = ssh[4] * xr * yr - ssh[5] * xr * yr
                syy = ssh[4] * yr * yr + ssh[5] * xr * xr
```

```
                szz, srz, wz, ur = ssh[0], ssh[1], ssh[2], ssh[3]
                for i, s in enumerate([sxx, sxy, syy, szz, srz, wz, ur]):
                    results[iz][i] += s
                iz += 1
        # top of lowest layer
        lz.append(len(th))
        z1.append(hns[-1] - th[-1])
        z2.append(hns[-1])
        ij0 = integrate_bessel(f_s_all, (7, ar, rr, hns[-1] - th[-1], th, ev, ps, len(th), hns))
        ssh = get_result((ij0, psi, rr))
        sxx = ssh[4] * xr * xr + ssh[5] * yr * yr
        sxy = ssh[4] * xr * yr - ssh[5] * xr * yr
        syy = ssh[4] * yr * yr + ssh[5] * xr * xr
        szz, srz, wz, ur = ssh[0], ssh[1], ssh[2], ssh[3]
        for i, s in enumerate([sxx, sxy, syy, szz, srz, wz, ur]):
            results[iz][i] += s
        iz += 1
    for ih in range(len(lz)):
        sxx, sxy, syy, szz, srz, wz, ur = results[ih]
        ss.append([lz[ih], z1[ih], z2[ih], sxx, sxy, syy, szz, srz, wz, ur])
    return ss
def print_results(ssa):
    print('n\tz1\tz2\tsxx\tsxy\tsyy\tszz\tsrz\twz\tur')
    for sa in ssa:
        print(sa)
    print()
def print_gels(params):
    print('Vehicle\tWheels\tOutput\tPavement')
    for param in params:
        print(param, '\t', end='')
    print()
    results = run_gels(params)
    print_results(results)
    return results
def hdes(vehicle, wheels, output, pavement, esubs, ths, layer_th, layer_str):
    results_str = []
    results_wz = []
    for th in ths:
        result_str = []
        result_wz = []
        for es in esubs:
            # set thickness
            pavement[0][layer_th - 1] = th
            # set subgrade modulus of elasticity
            pavement[1][-1] = es
```

```
        # set subgrade poisson's ratio
        pavement[2][-1] = poisson(es)

        results = run_gels((vehicle, wheels, output, pavement)) # quiet mode, else use print_gels

        # stress at bottom of layer_str at output location, max of sxx, syy
        result_str.append(max(results[(layer_str - 1) * 2 + 1][3], results[(layer_str - 1) * 2 + 1][5]))

        # surface deflection at output location
        result_wz.append(results[0][8])
    # save each result for all values of subgrade modulus of elasticity
    results_str.append(result_str)
    results_wz.append(result_wz)
return results_str, results_wz
```