taamak.h Documentation

Zeyuan Fang
fang.zeyuan@outlook.com

v1.0.0-rc

Contents

Introduction

taamak.h is a free/open-source C library for calculating responses in a truncated multi-layer linear-elastic pavement system. Its features include:

- Evaluation of stresses and displacements in a trimmed homogeneous linear elastic pavement model, subject to a square patch stress.
- Evaluation of surface deflection on a truncated multi-layered linear elastic pavement model, subject to a square patch stress.
- Back-calculation of Effective Slope Angle (ESA) of a homogeneous/multi-layered linear elastic pavement model, utilising FWD deflection measurements tested near the edge/discontinuity of the pavement system.

Info: Effective Slope Angle (ESA) is a novel entity in pavement engineering that quantifies the lateral support provided to the pavement system by the omitted region. For the scenario where a pavement system is abutting with a local soil medium, ESA represents the side support contributed by the local soil. In the scenario where a pavement system is experiencing significant cracking, ESA characterises the support offered by the intact material or through a stress-transfer mechanism across crack interfaces (or a combination of both). Under ideal conditions, ESA is 0° (representing a half-space model), while the most extreme scenario presents ESA at 90°.

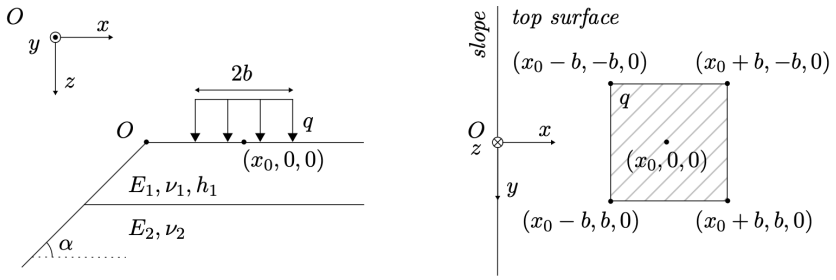
taamak.h is available on GitHub at https: //github.com/veslrs/taamak.h. It is distributed under the MIT License; users should note the additional notices and licensing terms for third party dependencies included with the repository.

`taamak.h` Tutorial

In this tutorial, we illustrate the usage of `taamak.h` via two trivial examples.

Example 1: Multi-layer near-edge deflections

As a first example, we'll look at the following simple problem calculating the deflections of a truncated two-layered linear-elastic pavement system, subject to a near edge patch stress.



Example 1 *(multi-layer near-edge deflections)*

Square stress patch of side $2b = 300\,\mathrm{mm}$ with uniform intensity $q = 0.5\,\mathrm{MPa}$ is applied to a trimmed two-layered elastic half-space. The elastic properties of Layer i $(i = 1, 2)$ are Young's modulus $E_1 = 500\,\mathrm{MPa}$, $E_2 = 50\,\mathrm{MPa}$ and Poisson ratio $\nu_1 = 0.3, \nu_2 = 0.4$; the corresponding layer thickness are $h_1 = 600\,\mathrm{mm}$, and a semi-infinite h_2. A Cartesian coordinate system placed at origin O: x-axis toward the patch, y-axis along the patch edge, and z-axis downward. The free side slope has an angle $\alpha = 75°$ with the top surface. In the top view the patch centroid lies at $x = x_0 = 300\,\mathrm{mm}$ and $y = z = 0$ with $x_0 > b$. We are asked to evaluate deflections at four points (in mm): $(0, 0, 0), (300, 0, 0), (600, 150, 0)$ and $(1200, 300, 0)$.

To implement the above example in C we would first do:

ex_01.c

```c
#define TMK_IMPLEMENTATION
#include "taamak.h"
```

to include the `taamak.h` header file as well as enabling the implementation. Then we would define our model object as:

ex_01.c

```c
tmk_model mdl;
tmk_init(&mdl, 2);
```

where the model `mdl` is initialised with two layers. For the specification of the model, we do:

ex_01.c

```
// set loadings
tmk_vec3 center = {300.0, 0.0, 0.0};
double b = 150.0;
double q = 0.5;
tmk_set_load(&mdl, center, b, q);

// set layer compositions
tmk_set_composition(&mdl, (double[]){0.0, 600.0});

// set layer properties
tmk_hs hs[] = {
    // layer #1
    {
        .youngs_modulus = 500.0,
        .poissons_ratio = 0.3,
    },
    // layer #2
    {
        .youngs_modulus = 50.0,
        .poissons_ratio = 0.4,
    },
};
tmk_set_material_properties(&mdl, hs);

// set slope angle
tmk_set_slope_angle(&mdl, 75.0);
```

There are two things to notice here. First, the function `tmk_set_composition` takes the depth of the *top interface* of each layer to specify the layer compositions. That does not affect the the application in a two-layered system, but for three layers and above, one should be careful with the calculation of the *top depths* from the given layer thicknesses. Second, one should check manually the alignment of the number of layers ad the number of material properties.
Now we set up evaluation points:

ex_01.c

```
tmk_vec3 pts[] = {
       {0.0, 0.0, 0.0},
       {300.0, 0.0, 0.0},
       {600.0, 150.0, 0.0},
       {1200.0, 300.0, 0.0},
};
tmk_set_evaluation_points(&mdl, 4, pts);
```

At this point, we can call `tmk_solve` to perform the evaluation, and lastly, we should call `tmk_checkout` to return evaluation status:

ex_01.c

```
// solve
tmk_solve(&mdl);

// done!
tmk_checkout(&mdl);
```

Assuming we save this in a file `ex_01.c`, we would compile and link (on Unix) with:

Command Line

```
$ cc ex_01.c -o ex_01 -lopenblas
```

The result of running the program should then be something like:

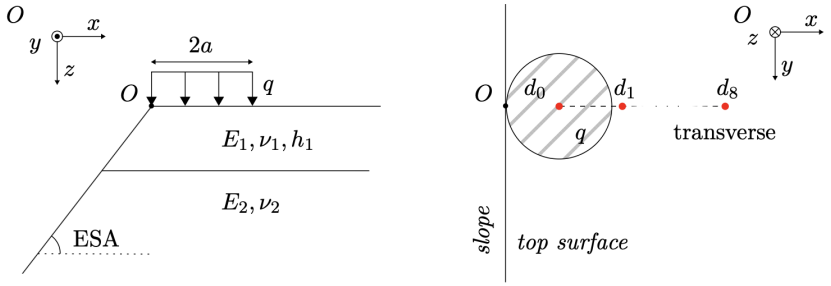
Command Line

```
x[mm]    y[mm]    deflection[micron]
0        0        651.5
300      0        810.8
600      150      521.0
1200     300      326.9
```

That is, it computed the deflections (in $\mu\mathrm{m}$) of given evaluation points.

Example 2: ESA back-calculation

In the second example, we'll look at the another simple problem back-calculating the Effective Slope Angle (ESA) of a two-layered linear-elastic pavement structure.



Example 2 *(ESA back-calculation)*

An FWD load plate with diameter $2a = 300\,\mathrm{mm}$ (simulated by square patch with $b = 133\,\mathrm{mm}$ to ensure same distribution area), exerts a uniform vertical stress with intensity $q = 0.707\,\mathrm{MPa}$. A two-layered elastic pavement structure. The load plate is placed near an edge of a pavement abutting local soil medium, or near the opening of a wide top-down crack, thus the pavement model was considered to be trimmed with an unknown ESA. A right-handed Cartesian coordinate system is introduced at the origin O, with the x-axis oriented perpendicular to the edge and pointing away from it, the y-axis coinciding with the edge, and the z-axis pointing downward into the medium. The cross-sectional view of the three-layered system. The elastic properties of Layer i $(i = 1, 2)$ are Young's modulus $E_1 = 500\,\mathrm{MPa}, E_2 = 80\,\mathrm{MPa}$ and Poisson ratio $\nu_1 = 0.3, \nu_2 = 0.4$; the corresponding layer thickness are $h_1 = 750\,\mathrm{mm}$, and a semi-infinite h_2. The load-plate is tangent to the edge, with the centre of the circle placed at $x = a$. The red markers, labelled $d_0, \ldots, d_8$, represent the sensors (geophones) of the FWD with transverse orientation, with x-offsets from the centre of the FWD load plate: $0, 200, 300, 450, 600, 900, 1200, 1500,$ and $1800\,\mathrm{mm}$, and the corresponding FWD deflection measurements: $0.659, 0.593, 0.404, 0.338, 0.287, 0.253, 0.201, 0.163, 0.134,$ and $0.113\,\mathrm{mm}$. We are asked to back-calculate the ESA.

To start with, we include the header file and implementation, then initialise and set up our model with available model specification parameters, same as Example 1:

ex_02.c

```c
#define TMK_IMPLEMENTATION
#include "taamak.h"

int main(void)
{
    // set loadings
    tmk_vec3 center = {150.0, 0.0, 0.0};
    double b = 133.0;
    double q = 0.707;
    tmk_set_load(&mdl, center, b, q);

    // set layer compositions
    tmk_set_composition(&mdl, (double[]){0.0, 750.0});

    // set layer properties
    tmk_hs hs[] = {
        // layer #1
        {
            .youngs_modulus = 500.0,
            .poissons_ratio = 0.3,
        },
        // layer #2
        {
            .youngs_modulus = 80.0,
            .poissons_ratio = 0.4,
        },
    };
    tmk_set_material_properties(&mdl, hs);

    // to be continued ...

}
```

Note that we do not initialise slope angle: it doesn't exist yet!
From here the set up diverges from Example 1: we now specify the FWD measurement coordinated and the corresponding data:

ex_02.c

```
// set FWD measurement points
int npts = 10;
tmk_vec3 pts[] = {
    {150.0, 0.0, 0.0},
    {250.0, 0.0, 0.0},
    {350.0, 0.0, 0.0},
    {450.0, 0.0, 0.0},
    {600.0, 0.0, 0.0},
    {750.0, 0.0, 0.0},
    {1050.0, 0.0, 0.0},
    {1350.0, 0.0, 0.0},
    {1650.0, 0.0, 0.0},
    {1950.0, 0.0, 0.0},
};
double vals[] = {
    0.659, 0.593, 0.404, 0.338, 0.287,
    0.253, 0.201, 0.163, 0.134, 0.113,
};
tmk_set_measurement(&mdl, npts, pts, vals);
```

Pretty straight-forward. Finally, same as Example 1, we can call `tmk_solve` to perform the ESA back-calculation, and call `tmk_checkout` to return evaluation status:

ex_02.c

```
// solve effective slope angle
tmk_solve(&mdl);

// done!
tmk_checkout(&mdl);
```

Assuming we save this in a file `ex_02.c`, we would compile and link (on Unix) with:

Command Line

```
$ cc ex_02.c -o ex_02 -lopenblas
```

The result of running the program should then be something like:

Command Line

```
[INFO] Iteration 11: trial algle = 62.2 [deg]. Evaluating.....
[INFO]   Signed difference with 62.2 [deg]: 0.000025
[INFO] Equivalent Slope Angle = 62.2 [deg]
```

That is, the remaining lateral support of the pavement structure can be quantified by an ESA = $62.2°$.

`taamak.h` Reference

`taamak.h` is a library, not a stand-alone program – it is designed to be called from your own program in C programming language. This reference section describes the programming interface (API) of `taamak.h`.

Compile and linking

An `taamak.h` program begins with including the header file. Note that by default only declarations are included, flag `TMK_IMPLEMENTATION` should be added before the header file to enable definitions of the functions.

demo.c

```c
#define TMK_IMPLEMENTATION
#include "taamak.h"
```

`taamak.h` depends on OpenBLAS (https://github.com/OpenMathLib/OpenBLAS) for numerical linear algebra functionalities. To compile the program, you will have to link it to the OpenBLAS library. You would normally link with a command like this:

Command Line

```
$ compiler -I<OpenBLAS include path> -L<OpenBLAS lib path> \
           -lopenblas -Wl,-rpath,<OpenBLAS lib path> input.c -o output
```

The `tmk_model` object

The `taamak.h` API revolves around an "object" of type `tmk_model` (a struct type). Via this object, all of the parameters of the pavement model are specified (number of layers, loadings, layer compositions, elastic properties, slope angle, etcetera), and then one finally passes this object to `tmk_solve` in order to perform the model evaluation. The object is initiated by passing it to:

```c
void tmk_init(tmk_model *mdl, const int n_layers);
```

which initiate `tmk_model` object in place, given the number of layers of the elastic pavement model, `n_layers`. When you are finished with the object, you should check out the program by calling:

```c
int  tmk_checkout(tmk_model *mdl);
```

which checks result flags within `tmk_model` and return success (0) or failure (1) to terminate the program.

Model specification

A general pavement-edge model requires the definition of several parameters (load, layer information, material properties, etcetera). Normally the first parameter to specify is loading, which is done by calling:

```c
void tmk_set_load(tmk_model *mdl, const tmk_vec3 center,
const double half_width, const double intensity);
```

where given the centre of the load patch (`center`, where `tmk_vec3` is a customised struct type representing three-dimensional vector/point), the half-width of the load patch b (`half_width`) and the intensity of the uniformly distributed load q (`intensity`), the loading information for the model will be initialised. Note that `tmk_vec3` is defined as:

```c
typedef struct {double x, y, z;} tmk_vec3;
```

To set elastic properties of the pavement model, you can pass the `tmk_model` object to the function:

```
void tmk_set_material_properties(tmk_model *mdl, const tmk_hs *hs);
```

where `tmk_hs` is the customised struct type for elastic properties of an individual layer, defined as:

```
typedef struct {double youngs_modulus, poissons_ratio;} tmk_hs;
```

By passing an array of `tmk_hs` containing all Young's moduli E_i and Poisson's ratio ν_i (for homogeneous model, such array has length 1), the material properties of all layers will be specified.
For multi-layered pavement structure, user should also setup layer thicknesses for the model. This is done by the function call:

```
void tmk_set_composition(tmk_model *mdl, const double *top_depths);
```

where `top_depths` is an array of all layers' top depths. For instance, given a three-layered system with layer thickness h_1, h_2 and a semi-infinite h_3, user should pass the array `(double[]){0.0, h_1, h_1 + h_2}` to this function.
From here, the setup of `tmk_model` diverges into two, depends on the use case of `taamak.h`. For the evaluation of surface deflection in a pavement system with known slope angle α, user should set up the coordinates of evaluation points and slope angle, before the final evaluation. This is done by the following function calls:

```
void tmk_set_evaluation_points(tmk_model *mdl,
const int npts, tmk_vec3 *pts);
void tmk_set_slope_angle(tmk_model *mdl, const double degree);
```

The first function, `tmk_set_evaluation_points`, accepts number of points (`npts`), and an array of type `tmk_vec3*`, representing coordinated of evaluation points. The second function, `tmk_set_slope_angle`, namely initialise slope angle for the model.
For the back-calculation of Effective Slope Angle (ESA), user should set up both FWD measurement array coordinates, and the corresponding measurement data. By calling function:

```
void tmk_set_measurement(tmk_model *mdl, const int npts,
tmk_vec3 *pts, double *vals);
```

user setup measurement data by passing number of FWD sensors (`npts`), array of sensor coordinates `pts`, and the corresponding array of deflection measurements `vals`.
Note that `taamak.h` offers a function to quickly setup a line with uniformly distributed points:

```
void tmk_linspace(tmk_vec3 *pts,
const tmk_vec3 *a, const tmk_vec3 *b, const int npts);
```

where user passes pre-allocated array of points (`pts`), alongside vertices at two ends `a` and `b`, and number of points to generate. The points will be distributes on the line segment defined by `a` and `b`, with the two end points also covered.

Perform the computation

Once all of the desired model parameters have been specified in a given object `mdl`, you can perform the computation by calling:

```
void tmk_solve(tmk_model *mdl);
```

where the function automatically detect the case to solve, perform the evaluation/back-calculation, then print the results to standard output.

Redefinable macros

`taamak.h` also provides a list of redefinable macros with default values. Experienced user could fine tune these parameters in order to reach the desired balance between numerical accuracy and computation efficiency. These macros include:

`TMKDEF`: Appends additional keyword to function declarations. Possible modification:

demo.c

```
#define TMKDEF static inline
```

`TMK_GEOM_M`: MFS collocation parameter, indicating number of points per side on simulated boundary (equivalent to $\sqrt{M}$ in this thesis). Default value is 50.

`TMK_GEOM_N`: MFS collocation parameter, indicating number of points per side on force boundary (equivalent to $\sqrt{N}$ in this thesis). Default value is 40.

`TMK_ELT_FACTOR`: factor used in the formulation of Equivalent Layer Thickness (ELT) theory. Default value is 0.9. Recommended range is 0.8 to 1.0.

`TMK_ELT_DEG_LIM`: Slope angle limit for multi-layered deflection evaluation. When a slope angle is set to be shallower than this limit, `tmk_solve` returns half-space solution. Default value is 10°.

`TMK_ESA_TOL`: Tolerance parameter in ESA back-calculation. Default value is 0.0001.

`TMK_ESA_ITER`: Maximum iterations for ESA back-calculation. Default value is 20.

`TMK_ESA_DEG_LIM`: Slope angle limit for ESA back-calculation. When a trial ESA is shallower than this limit, `tmk_solve` returns $\mathrm{ESA} = 0^\circ$. Default value is 10°.