Final Project
Due 4/29/2016 at **2 pm** on bCourses

1 Introduction

For the final project in E7, you will be tackling a real world problem to help a popular student group at Cal!

The University of California Marching Band performs at halftime during all home football games as well as some away games and bowl games (including Super Bowel 50!). As part of the performance, the band forms several formations on the field while marching and playing music (see Figures 1 and 2). In addition to practicing the music, the band members (the "marchers") spend a lot of time during rehearsals practicing the formations as well as the transitions from one formation to the next. For the show to be a success, each marcher must follow very specific instructions on where to stand in each formation, and how to get from one formation to the next.

Each formation is designed with a program called CalChart, written and maintained by Cal Band members. Currently, determining the transitions between each formation is the most time consuming and difficult part of the planning process. The planners must specify an exact route that each marcher (out of 180-190 marchers total) can traverse while playing their instrument, to get from their position in one formation to their position in the next formation. Furthermore, each marcher must be able to make the transition in a specified number of steps (corresponding to the number of beats in the music they are playing). Lastly and most importantly, no two marchers can collide during these transitions.

For your final project, your task is to write a program that can coordinate a transition strategy between one formation and the next. Currently, these transitions are done manually. For a single transition with 180 marchers, it can take Cal Band students 2-10 hours to do this manually. There are 6-8 transitions like this per show, so this adds up quickly. The challenge is to automate this process to save time, optimizing the band member's movements and thus enabling more complicated designs. This will make the band performances easier to coordinate and design, and students and alumni will continue to cheer, "Cal Band Great"!

1.1 Formations and transitions

The marching band performs its shows in an area that we will call the "field". We divide the field into a two-dimensional grid that has n_r rows and n_c columns. The locations of the marchers in the field are described as their locations in the grid. There are therefore $n_r \times n_c$ possible locations where marchers can be at a given time.

Figure 1: Cal Band in script Cal formation

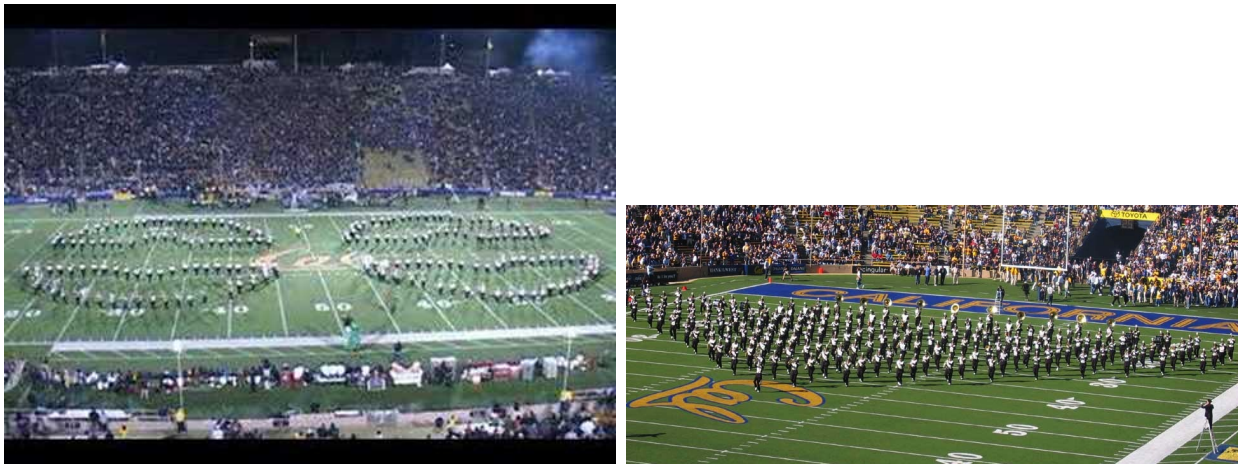

Figure 2: More examples of band formations

For each transition, we define:

- The number n_b of band members (marchers) involved in the transition.
- The initial formation: the locations of the n_b marchers before the transition.
- The target formation: the locations that must be occupied by the n_b marchers after the transition.

Each marcher is identified by a unique **tag**, an integer that can range from 1 to n_b. We represent the initial formation in Matlab as an $n_r \times n_c$ array of doubles, which we call the **initial formation array**. In the initial formation array, a zero indicates that there is no marcher at that location, and a non-zero value indicates the tag of the marcher that is located there. As a consequence, each integer in the range 1 to n_b (included) appears once and exactly once in the initial formation array. All the other values in the initial formation array are 0.

We also represent the target formation in Matlab as an $n_r \times n_c$ array of doubles, which we call the **target formation array**. In the target formation array, a 0 indicates that there should not be any marcher at that location in the target formation, while a 1 indicates that there should be a marcher at that location in the target formation. Exactly n_b elements of the target formation array will have a value of 1 (one for each marcher). All the other elements of the target formation array will have a value of 0. Note that we do not initially specify which particular marcher should be located at which location in the target formation. Your function should decide which target location each marcher should go to in order to properly complete the transition. There will likely be many possible different ways to complete a given transition and many possible final configurations of marchers that match the target formation!

Marchers move between formations following the beat of the music they play. When they move, they move at the speed of one grid cell per 2 beats. If you are curious how this relates to a real football field, a grid cell is 2 steps long (one step with the left foot, one step with the right foot), and there are 8 steps to each 5 yards. So in total, the length of the field is 84 steps, or 42 grid cells, and the width of the field is 160 steps, or 80 grid cells. For your purposes, what you need to know is that in two beats, a marcher can move to the location in an adjacent row or adjacent column on the grid. For this assignment, marchers cannot move on a diagonal, so only motions in the x or y directions are allowed. Following this, marchers can reach their target location from their initial location in one of three ways:

- If their target location is the same as their initial location, then they must stay in place for the duration of the transition.
- If their target location is in the same row or the same column as their initial location (but the initial and target locations are not the same), then they must move in a straight line.
- In any other case, they must move in a straight line, followed by a 90-degree turn, followed by a straight line.

We will use the cardinal directions North (N), South (S), East (E), and West (W) to describe which directions marchers can move in the grid:

- Moving North means moving along a given row, in the direction of increasing column indices.

- Moving South means moving along a given row, in the direction of decreasing column indices.
- Moving East means moving along a given column, in the direction of increasing row indices.
- Moving West means moving along a given column, in the direction of decreasing row indices.

Pay careful attention to the orientation of the field (see Figure 3 later) as we have defined the directions the same way they are defined in CalChart.

For a given transition, certain marchers can be instructed to wait for a certain number of beats before they start moving away from their initial location. Having certain marchers wait before moving can be useful for avoiding collisions between marchers. However, once a marcher starts moving, they move at a constant speed until they reach their target location. In other words, after a marcher starts moving, they do not interrupt their movement before reaching their target location. If a marcher reaches their target location before the end of the transition, then they simply wait at their target location until the transition is completed.

1.2 Your task

You are tasked with writing a computer program that determines the paths that each marcher should take to perform transitions between different formations. Your main function should use the following header:

```
function [instructions] = calband_transition(initial_formation, ...
                                             target_formation, max_beats)
```

where `initial_formation` is the initial formation array, `target_formation` is the target formation array, and `max_beats`, a strictly positive **even** integer of class double, is the number of beats within which the transition must be completed. The Cal Band uses 32 beats for most of their transitions. Some shorter transitions are sometimes completed in 16 beats. In any case, the Cal Band prefers to use a number of beats for the transition that is a multiple of the length of the bar (most often 4 beats) of the music they are playing while doing the transition. Note that your function should work for **any** value of `max_beats` for which the transition is possible. In other words, your function should **not** assume any specific value for `max_beats`. The formats of the initial formation array and of the target formation array were described above in Section 1.1.

The output `instructions` is a 1 by n_b `struct` array, where n_b is the number of marchers involved in the transition. The struct array `instructions` describes the instructions given to the marchers in order for them to complete the transition. More precisely, the information in the element of `instructions` that has index i_b specifies the instructions given to the marcher

who has the tag i_b. For example, `instructions(7)` specifies the instructions given to the marcher who has the tag 7. The struct array `instructions` must have the following fields, and only these fields:

- `i_target`: row index of the target location assigned to the marcher. It should be a strictly positive scalar integer of class `double` with $1 \leq$ `i_target` $\leq n_r$.
- `j_target`: column index of the target location assigned to the marcher. It should be a strictly positive scalar integer of class `double` with $1 \leq$ `j_target` $\leq n_c$.
- `wait`: number of beats that the marcher will wait before moving. It should be a positive (or zero) **even** scalar integer of class `double` with $0 \leq$ `wait` $\leq$ `max_beats`.
- `direction`: direction in which the marcher will move to reach their assigned target location. It should be a character string of length 1 or 2. Possible values are the following strings (without quotes) '.', 'N', 'S', 'E', 'W', 'NE', 'EN', 'NW', 'WN', 'SE', 'ES', 'SW', and 'WS'. The value should be '.' if and only if the target location assigned to the marcher is the same as their initial location. The value should be 'N' if the marcher must only move North to reach their target location, the value should be 'NE' if the marcher must first move North (all the way to the column where their target location is), then move East to reach their target location. The value should be 'EN' if the marcher must first move East (all the way to the row where their target location is), then move North to reach their target location, and so on and so forth.

Additional details:

- If at a given time, two or more marchers are located in the same grid cell, this is considered a collision.
- Also, marchers collide if they switch grid cell locations over the course of one timestep (since that means they are moving toward one another, and would collide as they try to switch places).
- When grading your function, the initial formation array and the target formation array that will be passed as input arguments to your function will follow the formats described in Section 1.1. Similarly, the value passed to your function as the input argument `max_beats` will be a strictly positive even integer of class double. We will **not** test your function with input data that do not follow these rules, so your function does **not** need to check for the correctness of the inputs.

2 The transition visualizer

You can use the function `calband_visualize_transition_v01`, available on bCourses, to visualize a transition. `calband_visualize_transition_v01` has the following header:

```
function [] = calband_visualize_transition_v01(initial_formation, ...
                            target_formation, instructions, max_beats)
```

where `initial_formation`, `target_formation`, `instructions`, and `max_beats` are the same as in the header of the function `calband_transition`. Calling this function should open a figure, which we will call the visualizer. The visualizer (see example in Figure 3) shows the grid, the locations of the initial positions of the marchers as blue non-shaded squares, the locations of the target locations as green shaded squares with rounded corners, and the current locations of the marchers as numbers. You can interact with the figure using keyboard shortcuts (see Table 1). For example, press the right-arrow key to visualize the locations of the marchers after the first two beats of the music have been played. You can press the "h" key to show (in the command window) a list of all the keyboard shortcuts that you can use to interact with the visualizer. Some of the features implemented in the visualizer are:

- The visualizer will only be generated if the following input arguments are valid: `initial_formation`, `final_formation`, or `max_beats`.
- The visualizer will only be generated if the input argument `instructions` is a $1 \times n_b$ struct array which has the four required fields: `i_target`, `j_target`, `wait`, and `direction`.
- Each marcher will move in the visualizer only if the instructions given to them follow the format described in this document.
- Each marcher will move in the visualizer only if following their instructions take them closer to their assigned target location (not further away).
- The visualizer will display warnings (in the command window) if certain conditions are met, for example if collisions occur.

Table 1: Keyboard shortcuts that can be used to interact with the visualizer. When several keys lead to the same effect, these keys are listed as a comma-separated list.

Key	Effect
right arrow, down arrow	Next frame
left arrow, up arrow	Previous frame
home, page up	Initial frame
end, page down	Final frame
escape, q	Close visualizer
enter	Run transition forward
backspace	Run transition backward
f	Increase visualizer frame rate
s	Decrease visualizer frame rate
h	Show the list of shortcuts in the command window

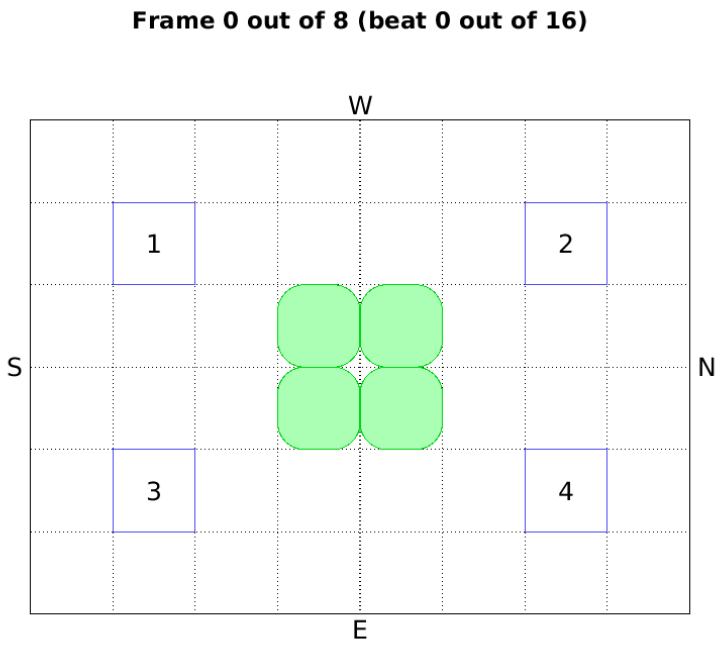

Figure 3: The visualizer showing the initial frame of the corners_to_center transition (see Table 2). The visualizer shows the grid, the locations of the initial positions of the marchers as blue non-shaded squares, the locations of the target locations as green shaded squares with rounded corners, and the current locations of the marchers as numbers.

If you find a bug in the visualizer, please report it by posting on Piazza following the instructions described below:

- Describe the input data that you used when calling the visualizer function when the bug happened. Explain how the visualizer should have behaved given the input data you used, and how the visualizer did actually behave.
- If possible, provide one or more screenshot(s) documenting what happened.
- Optional: propose a solution describing how the bug can be fixed.

3 Project deliverables

For this project you will work in a team of 2 or 3 students (groups of 3 are highly preferred). Your team must consist of students in your lab group. You are allowed to write code together for this and only this assignment. You should share code only within your team, and not with other students in the class.

3.1 Beta test

Beta testing is an early release of software to friendly users for testing. After spending some time working on the project, you will submit your current version of your code to your GSI for testing and feedback. The only requirement for receiving full credit for the beta test is turning your code on time and having that code return without throwing an error (even if the marchers are not doing the right thing inside your function). In your submission you can include questions for your GSI so you can get some feedback. Work hard on the project before the beta test, because the more you turn in by this deadline, the more feedback you can get from your GSI.

3.2 Final code - structure and comments

Before starting to write code, you should outline an appropriate code structure and maintain a consistent line of communication with your teammates. There will be a group created for you inside bCourses where you can collaborate with your team members. Make sure to outline the various sections of code you will need to develop to accomplish the project as a whole, as this will help you tackle the project as a team. Needless to say, comments within your code will become essential for conveying its meaning to the other members of your group. Because of this, you will be graded on your overall code structure and comments. These should be easy points, but they are here to encourage you to use strong coding practices that will help you save time in the long run, especially when collaborating with others.

3.3 Final code - basic requirements

First and foremost, your code should match the format provided in this document. That is to say, you should be able to get output from your function that will successfully run in the visualizer without causing an error. Note: meeting these requirements, even if none of the marchers moves at all (!), is your first priority, and will ensure that you earn points from this section. Doing so will also enable you to earn points from further testing.

Here is a checklist of these basic requirements for your function:

- Your function must complete without throwing errors.
- Your program should return a result in less than 2-3 minutes, which should be adequate time to perform the required basic functionality. Your program may take longer time to perform additional/advanced functionality, but this should be described in your writeup. In other words, if your program takes longer to compute, your writeup (see below) should explain and justify why. The grading will be done on the computers of the Etcheverry E7 computer lab. We also recommend that you do not use a `while` loop anywhere in your code. If your code enters an infinite `while` loop when tested on

a particular test case, then your function will not meet the basic requirements for that test case.

- The output of your function should be a $1 \times n_b$ struct array.
- The list of fields in your output struct array should be `'i_target'`, `'j_target'`, `'wait'`, `'direction'`. Note that the order of the fields in your output struct array does **not** matter.

3.4 Final code - intermediate requirements

In addition to your code successfully matching the format required by the visualizer, you will earn additional points if the result within the fields of your `instructions` structure make sense.

The following is a checklist of the intermediate requirements:

- For each marcher, the field `wait` must be valid (see Section 1.2).
- Each marcher should be assigned a valid target location (see Section 1.2). No two marchers should be assigned the same target location.
- Each marcher should be assigned a valid moving direction (see Section 1.2).

3.5 Final code - advanced requirements (transitions)

Your function will be tested on transitions of varying complexity. For each transition, you will be given a score based on whether your function can produce a correct instruction structure.

Your grade for this section will be based on the following criteria:

- The target formation is correctly completed at the end of the transition, meaning that the marchers actually move to the proper locations as given by your instructions, and all the target locations are occupied by a marcher.
- No collisions happen during the transition.

We provide you with a variety of transitions that you can use to test your function. Each of these transitions can be achieved in 32 beats. Each transition is made of an initial formation array and a target formation array. These transitions are provided in the file `sample_transitions.mat` available on bCourses. You can load the content of this file into your workspace using the `load` function. Table 2 shows all the sample transitions available

Table 2: Summary of the sample transitions provided to you. n_b is the number of marchers involved in the transition, n_r is the number of rows in the grid, and n_c is the number of columns in the grid.

Name of the transition	Size	n_b	$n_r \times n_c$
block_splits	tiny	4	6×8
block_regroups	tiny	4	6×8
block_traverses	tiny	4	6×8
corners_to_center	tiny	4	6×8
line_to_uparrow	small	20	12×22
blocks_to_circles	small	20	7×17
blocks_approach	small	20	7×13
e7_to_happy	medium	40	13×18
triangle_to_box	medium	40	23×23
go_bears	large	60	19×42
script_cal	huge	80	16×21
bridge_to_train	fullscale	180	27×49

in `sample_transitions.mat`. The names of the initial and target formation arrays for each transition are constructed as follows:

`<size>_<name>_<initial or target>`

where `<size>` and `<name>` are the size and the name of the transition as given by Table 2 and `<initial or target>` is `initial` for the initial formation array and `target` for the target formation array. For example, the initial and target formation arrays that correspond to the first transition listed in Table 2 are `tiny_block_splits_initial` and `tiny_block_splits_target`, respectively. Note that while the band usually works with a 42×80 grid, we cropped the sample transitions to smaller grids for this project.

Additional sample transitions will be tested during the grading process. You can also make up your own transitions. Note that the larger transitions (sizes "large" and above) may be difficult to solve. They are provided along with the smaller transitions for you to be able to test your function on transitions of varying complexity.

We **strongly** recommend that you prioritize your efforts on having your function satisfy the items on the "basic requirements" checklist for all test cases (both those provided to you and those you design yourself). Once your function complies with the basic requirements, work on achieving the "intermediate requirements" checklist before working on correctly completing the transitions. It is also suggested that you aim to get your function to work with the smaller test cases before moving on to larger formations.

Note that for the grading, the marchers will behave as they behave in the visualizer. In particular, as noted above, the marchers will only move in the visualizer if the instructions move them toward their assigned target location.

3.6 Final writeup

Your final project submission will include a 1-2 page executive summary of your final submission, which includes an overview of your algorithm and how it works, and why you made the design decisions you did. Think of this like an advertisement for your code, highlighting the key features and making it easy for your customer (the Cal Band) to understand what you did. Include a title and your team member names at the top of your summary, and make proper use of headings, sections, figures, as needed to support the text.

Your writeup will be graded by your GSIs based on the use of appropriate technical writing and the adequacy of your description (we should be able to to understand your methodology without looking at your code). Also make sure you are within the page limit to receive full credit.

3.7 Grading breakdown

You will be scored out of 100 points as a group as follows:

- Beta test - 15 points
- Final code - 70 points
  - Structure and comments - 5 points
  - Basic requirements - 25 points
  - Intermediate requirements - 20 points
  - Advanced requirements (transitions) - 20 points
- Final writeup - 15 points

Group member grades may be affected by peer evaluations, which you will submit online through bCourses when your final project is submitted. Ideally, all team members will pull an equal weight, contributing where each person has greatest strengths and combining your skills to accomplish more. If there are problems with your group, please let your GSI know AS SOON AS POSSIBLE. You can also use the peer evaluation as a way to indicate any concerns you had during the final project.

The grading of your function will be done on the computers of the Etcheverry E7 computer lab. You should **not** use functions or other Matlab functionality that are **not** part of Matlab's base installation if they are not already installed on the computers of the Etcheverry E7 computer lab.

4 Tips and tricks

4.1 Things to keep in mind

- Be creative! This is an open-ended project, so there is no right or wrong way to code the transitions.
- Brainstorm! Spend time with your team talking about different approaches you could take. Think about all the concepts you have learned in class and how you might apply each of them to your project, e.g. iteration, recursion, branching, regression, root finding, series, interpolation, integration, Monte Carlo approaches, etc. You never know what might trigger an idea for a new algorithm.
- Be inspired by the real Cal Band (watch videos!) and think about which elements of the human decision making process (when the transitions are charted by hand) that you can automate.
- Always put basic functionality first, ahead of fancy features (for example, make sure your function meets the basic and intermediate requirements described above before worrying about avoiding collisions).
- Create a modular program so you can divide the project into pieces that can be tested individually and that each team member can work on.
- Use functions and subfunctions to organize your program in a logical way. Test the pieces separately before putting them all together. This will help with debugging!
- Needless to say, your code should be well commented. This will facilitate collaboration among all team members.
- We encourage you to make up your own transitions to test how your function performs in a variety of situations.
- Beautiful transition algorithms will be used by the Band!! Projects which demonstrate the best transitions will get to present their results to the Cal Band, which may adopt your strategy for future performances. Who knows, you could get famous and end up in the Super Bowl like the Cal Band!
- Realize that your code may not be as good as a human at making the transitions. There may be some transitions where your code fails and a human would have to intervene to fix it. Even small improvements to the manual process will be a benefit to the band, so the goal is to do your best! Strategize about how to design your code to meet the basic and intermediate requirements first, then go as far as you can go with it!
- Yes, this is an open-ended project and it is hard. But everything feels impossible until it actually becomes possible. Rest assured that we know this is hard and will be

reasonable with the grading. Follow the guidelines above, and don't forget to have fun with this!!

4.2 Extra information

- The Cal Band's website is here.
- See a collection of videos of Cal Band performances here.
- Animated transitions from the Fall 2015 season can be viewed online using the CalChart Viewer, where you can step through transitions between formations. Notice that occasionally marchers move on a diagonal, but you do not need to consider this in your code, as the majority of the transitions are done on the grid.
- The source code for CalChart is here if you are curious.
- Some Cal Band history about charting: alumni band website.

5 Submission instructions and important dates

Your function header and file name must appear *exactly* as they do in the guidelines above. So, make sure the variables have the right capitalization, the functions have the correct name, and the inputs and outputs are in the correct order.

Note that minor updates to the project guidelines may be posted at a later date - it is your responsibility to follow up and read announcements related to the project.

5.1 Beta test - due April 4 at 2pm on bCourses (2 hr grace period until 4pm)

This is an easy way to earn points for the project, so double check that your code does not throw any errors when you plug in the sample cases before you submit it. Also remember that this is a great opportunity to get feedback if you are stuck on the problem. See discussion above. You can submit questions or notes to your GSI through bCourses when you upload your file.

Your project group will have a single submission in bCourses. You must submit a zip file called `beta.zip` that contains:

- A file `calband_transition.m` that contains your function `calband_transition` as the main function
- (Optional) Any other .m files needed by your main function.

5.2 Final submission - due April 29 at 2pm on bCourses (2 hr grace period until 4pm)

Make sure to get it uploaded early, well before the deadline! Go over at least the basic and intermediate code requirements, the simple test cases, and make sure your code is well formatted and commented.

Your project group will have a single submission in bCourses. You must submit a zip file called `project.zip` that contains:

- Your write-up **in pdf format**;
- A file `calband_transition.m` that contains your function `calband_transition` as the main function
- (Optional) Any other .m files needed by your main function.

5.3 Project showcase - Monday May 2, 2-3 pm (during lecture)

We will celebrate what we have learned during the project in addition to doing a review for the final exam, both during lecture.

Best of luck to you all! Remember to have fun with this!
GO BEARS!