Introduction to `ZPDGrowthTrajectories`

The `ZPDGrowthTrajectories` package contains functions and tools for exploring the behavior of a quantitative theory of academic achievement growth based on Vygotsky's Zone of Proximal Development (ZPD). The details of this theory will be presented in an upcoming publication by Matthew T. McBee, D. Betsy McCoach, and Matthew C. Makel.

This vignette illustrates how to use the package.

Installing the package

The package is available on the lead author's github site. To install the package, you will first need to download the `devtools` package. You will also need the `knitr` and `rmarkdown` packages in order to build the vignette.

```
install.packages("devtools")
```

After installing `devtools`, you can use the `install_github()` function to install the package.

```
devtools::install_github("mcbeem/ZPDGrowthTrajectories", build_vignettes=TRUE)
```

Once the package is installed, which only needs to be done once, you can activate the package's functions using `library()` or `require()`.

```
require(ZPDGrowthTrajectories)
```

Description

The package consists of two principal functions, `ZPDGrowthTrajectories()` and `ZPDGrowthTrajectories_multicore()`. Both produce equivalent results. The *multicore* version of the package provides a considerable performance boost when running large-n simulations so long as your computer has appropriate hardware. However, for small-n simulations, the single-core version of the function runs slightly faster to due to avoiding extra overhead.

The number of processors can be displayed by running `parallel::detectCores()` at the console.

```
parallel::detectCores()
```

```
## [1] 4
```

Background

This theoretical model represents academic achievement as a student's position along a number line bounded by zero and one. The student's zone of proximal development (ZPD) is a function of their achievement and is modeled by a normal curve that is attached or 'yoked' to the student's current level of achievement. As the student's achievement changes, their ZPD changes with it.

The achievement growth rate (or learning) during a period t depends on how well the student's ZPD aligns with the instruction to which they are exposed. Sub-optimal growth occurs when the curriculum is above the bulk of the student's ZPD ("too hard") or below it ("too easy").

The core dynamic of the this theoretical model is the the student's achievement growth rate is proportional to the overlap between their ZPD and the instruction to which they are exposed, both in school and at home.

Using the Package

Using the package to create synthetic growth trajectories requires the specification of the following information:

- **Student characteristics**: for each student, the *learning rate*, *decay rate* (forgetting speed), *initial achievement*, and the quality of the *home environment*. There is no intrinsic scaling for these values, currently, and the meaning of a particular value depends on several other arguments to the `ZPDGrowthTrajectories` function. However, they should all be positive.
- Parameters controlling the **zone of proximal development**. These control the width and location of each student's zone of proximal development relative to their current achievement level.
- **School curriculum**: for each grade level that the student will encounter, the achievement level of the content to be presented must be described. This includes the lower and upper values of achievement for which the curriculum reaches maximal intensity as well as parameters describing how rapidly the intensity of the curricular content falls both below the lower bound (e.g., review of content from prior grades) and above the upper bound (e.g., above grade-level content).
- **Home curriculum**: students learn outside of school. The home curriculum function describes the intensity of home instruction at different levels of achievement. This is typically a decaying function, as children typically experience much more intense instruction in early childhood content at home (e.g., colors, animal sounds, learning letters and numbers) than in advanced content (e.g, Pythagorean theorem, mitochondria, properties of gerunds).

In addition, a set of **control parameters** are specified. These typically have reasonable defaults. The control parameters include `integration.points`, `zpd.scale`, `decay.weight`, `use.GPU`, and `verbose`.

- `integration.points`: A positive integer. It controls the tradeoff between precision and execution speed. Typically, more than a few hundred integration points yields only a tiny benefit to precision. Defaults to 200.
- `zpd.scale`: A positive number. A global parameter for adjusting the growth rate. Defaults to 0.055. Reasonable values are usually quite small (<0.10).
- `decay.weight`: A positive number. A global parameter for adjusting the decay (forgetting) rate. Defaults to 0.005. Reasonable values are usually quite small (<.02).
- `useGPU`: A logical value. Should the matrix computations be executed on the GPU? In my tests this has no benefit and dramatically slows execution for small simulations, but I have not been able to test this with a powerful GPU. Defaults to `FALSE`. It is possible that, with appropriate hardware, this option could provide increased execution speed for large simulations.
- `verbose`: A logical value. Should status reports be printed to the console during execution? Note that fewer updates are possible when using the *multicore* version of `ZPDGrowthTrajectories` due to the nature of parallel computing. Defaults to `TRUE`.

Student characteristics

The following code demonstrates how characteristics for four simulated students can be defined. These objects are vectors created by the concatenate `c()` function. The length of these vectors defines the number of students to be simulated. They must all be the same length.

```
# generate data for four students

# learning rate
learning.rates <- c(.08, .10, .12, .18)

# decay rate
decay.rates <- c(.04, .03, .02, .01)

# initial achievement
initial.achievements <- rep(0, times=4)

# quality of home environment
home.environments <- c(.06, .12, .15, .20)
```

Each student has a unique position in the student characteristics vectors. For example, student #1's values are found in the first position of each of these vectors. His learning rate is 0.08, decay rate is 0.04, initial achievement is 0, and home environment is 0.06.

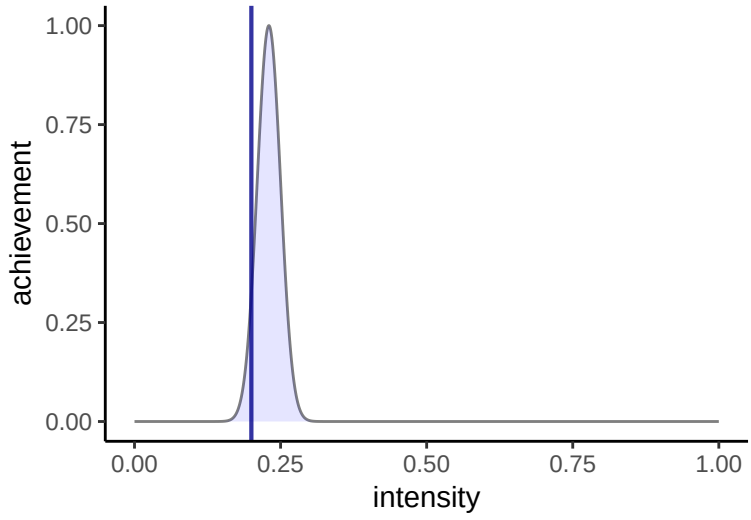
The Zone of Proximal Development (ZPD)

The following code provides plots a student's current achievement (vertical red line) and ZPD. The 'width' of the ZPD is controlled by the value of `zpd.sd`, its position with respect to current achievement is controlled by the value of `zpd.offset`. Both of these are arguments to the `ZPDGrowthTrajectories()` function.

Note that the ZPD is normalized to have a maximum value of 1.0 instead of an area of 1.0. This is a feature of the `ZPDGrowthTrajectories()` function which allows users to alter the width (standard deviation) of the curve without simultaneously altering its height.

The `visualizeZPD()` function produces plots to assist in exploring the consequences of different choices of `zpd.sd` and `zpd.offset`.

```
visualizeZPD(achievement=.2, zpd.sd=.02, zpd.offset=.03)
```



Defining the School Curriculum

Assignment Vector

Now that the characteristics of the four students has been specified, the *assignment vector* will be created, in this case representing 1,300 units of time that we will think of as days. All four students will begin in the same place, zero achievement. For the first 800 days the students are at home. From day 800-1,000 they go to school (let's call this 'kindergarten'). From day 1,001 to 1,100 there is a summer break. And from day 1,101 to day 1,300 the students are back in school (first grade).

To begin, we will create an object that describes what curriculum the students are exposed during each time period. This object is a vector whose length is the number of days. A value of zero represents times of no school exposure. Other numbers represent specific grades, and the description of the curricula to be presented in those grades is found in the corresponding rows of the school curriculum matrices. So a value of "1" in a specific position ("day") of the assignment vector means that the students will be exposed to the school curriculum described in row 1 of the school curriculum description matrices, arguments

```
assignment <- c(rep(0, times=800), rep(1, times=200),
                rep(0, times=100), rep(2, times=200))
```

The theoretical model considers two sources of instruction: what is learned at home, and what is learned at school. During time units (or "days") when the student is not in school (in other words, when there is a zero in the assignment vector), the student is exposed only to the "home curriculum." When the student is in school, their learning is based on a mixture of home and school exposure. The ratio of these exposures can be controlled by the `dosage` argument to the `ZPDGrowthTrajectories()` function. For example, when `dosage=.5`, the student's growth during academic years is 50% a function of the school curriculum and 50% a function of the home curriculum.

School Curriculum

The academic content ("curriculum") for each grade is co-located on the same scale as academic achievement.

Next, the two objects necessary for describing the school curriculum are created. These objects are matrices where the number of rows is equal to the number of different curricula (e.g., kindergarten, first grade, and so on), and the number of columns is equal to the number of different *versions* of each grade-level curriculum that will be presented to students (e.g., typical grade-level, remedial, or advanced). When more than one version of each grade-level curriculum is available, each student receives the most beneficial curriculum for each simulated day.

Note that these object *must* be explicitly created as matrices using the `matrix()` function.

For this example, there is only one version of each of the two grade level curricula. Therefore this matrix has dimensions 2×1.

```
curriculum.start.points <- matrix(c(.10, .15), nrow=2, ncol=1)
```

Thus, the kindergarten curriculum reaches full intensity at the achievement level of 0.1, while the first grade curriculum reaches full intensity at the achievement level of 0.15.

Now the functioning of the `assignment` object should be more clear. Non-zero values in the `assignment` vector refer to rows in the `curriculum.start.points` object created above. For example, on days 801 through 1,000, students are exposed to the school curriculum described in row 1 of this curriculum matrix (hence the value of "1" in the assignment matrix for these days). On days 1,101 through 1,300, the value of "2" in the assignment matrix refers to the 2nd row of `curriculum.start.points`.

The curriculum width matrix describes the *span* of the curriculum. The upper bound of the full intensity of the school curriculum is located at the starting point plus the width.

```
curriculum.widths <- matrix(c(.05, .05), nrow=2, ncol=1)
```

Based on the values assigned to `curriculum.start.points` and `curriculum.widths`, the Kindergarten curriculum here begins at the achievement level of 0.1 and ends at $0.1 + 0.05 = 0.15$. The first grade curriculum begins at 0.15 and ends at 0.2.

Visualizing the School Curriculum

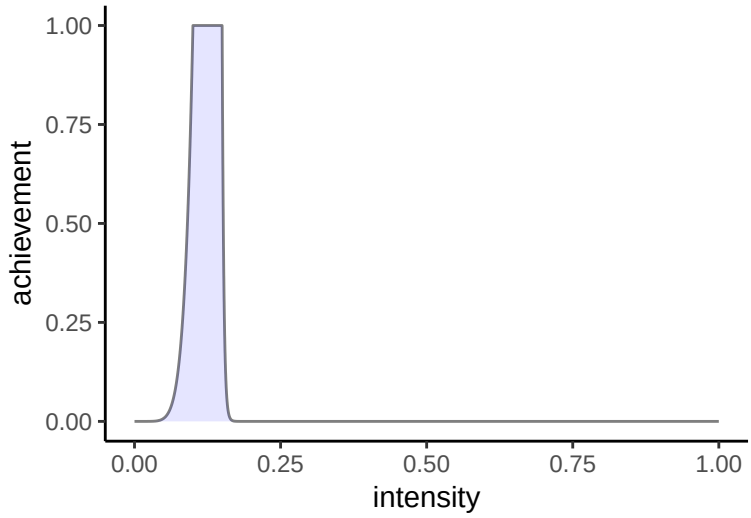
The `ZPDGrowthTrajectories` package uses the generalized trapezoidal distribution to model the school curriculum. The shape of this distribution is governed by five parameters, allowing the user to specify flexible shapes. The trapezoidal distribution function is `trapezoid::dtrapezoid()`. The `curriculum.start.points` and `curriculum.widths` arguments have already been discussed.

The `curriculum.lower.slope` and `curriculum.upper.slope` arguments describe how rapidly the curricular intensity fades for achievement levels below the lower bound of full intensity (e.g., review) and above the upper bound (e.g., advanced content). In general, values between 1-10 describe slow fade-out while values in the hundreds specify quick fade-out.

The `alpha` argument describes the shape of the peak intensity. The default setting of `alpha=1` means that the curricular intensity is constant ("flat") between the lower and upper bounds.

The best way to develop an intuition about these values is the experiment with the code for plotting the school curriculum function. The `visualizeSchoolCurriculum()` function facilitates this exploration. This function plots the school curriculum function for different values of its five arguments.

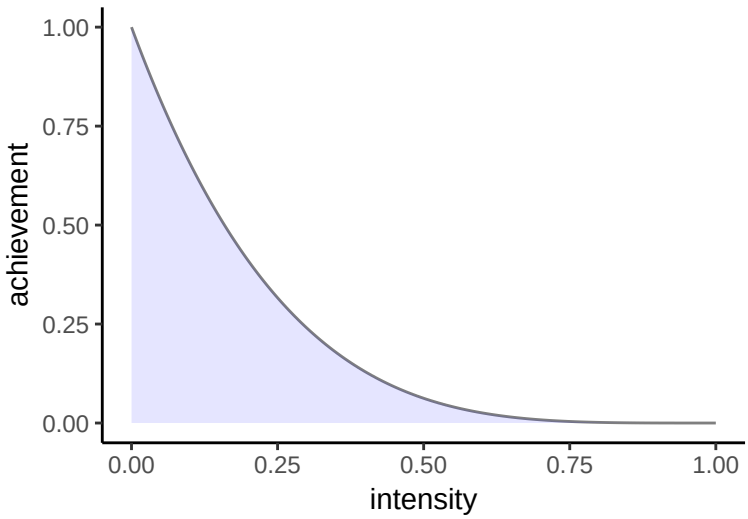
```
visualizeSchoolCurriculum(start.point=.10, width=.05,
                          lower.slope=8, upper.slope=300, alpha=1)
```



Home Curriculum

The home curriculum describes learning outside of school. Based on a beta distribution, the default values for the `home.curriculum.shape1` and `home.curriculum.shape2` arguments (which are the `shape1` and `shape2` argument to `dbeta()`) describe the following shape to the home curriculum. The `visualizeHomeCurriculum()` function assists in visualizing the consequences of different values of the `home.curriculum.shape1` and `home.curriculum.shape2` arguments to `ZPDGrowthTrajectories()`.

```
visualizeHomeCurriculum(shape1=1, shape2=5)
```



With these parameters set, the `ZPDGrowthTraejectories()` function can be used to generate synthetic growth trajectories for the four students.

```
trajectories <- ZPDGrowthTrajectories(
  output.format="wide",
  days=1300,
  assignment=assignment,
  curriculum.start.points=curriculum.start.points,
  curriculum.widths=curriculum.widths,
  dosage=.8,
  learning.rates=learning.rates,
  decay.rates=decay.rates,
  initial.achievements=initial.achievements,
  home.environments=home.environments,
  integration.points=200,
  curriculum.lower.slope=8,
  curriculum.upper.slope=300,
  alpha=1,
  home.curriculum.shape1=1,
  home.curriculum.shape2=5,
  zpd.offset=.03,
  zpd.sd=.028,
  zpd.scale=.15,
  decay.weight=.005,
  useGPU=FALSE,
  verbose=FALSE)
```

The first few rows of the `trajectories` object can be examined using the `head()` function.

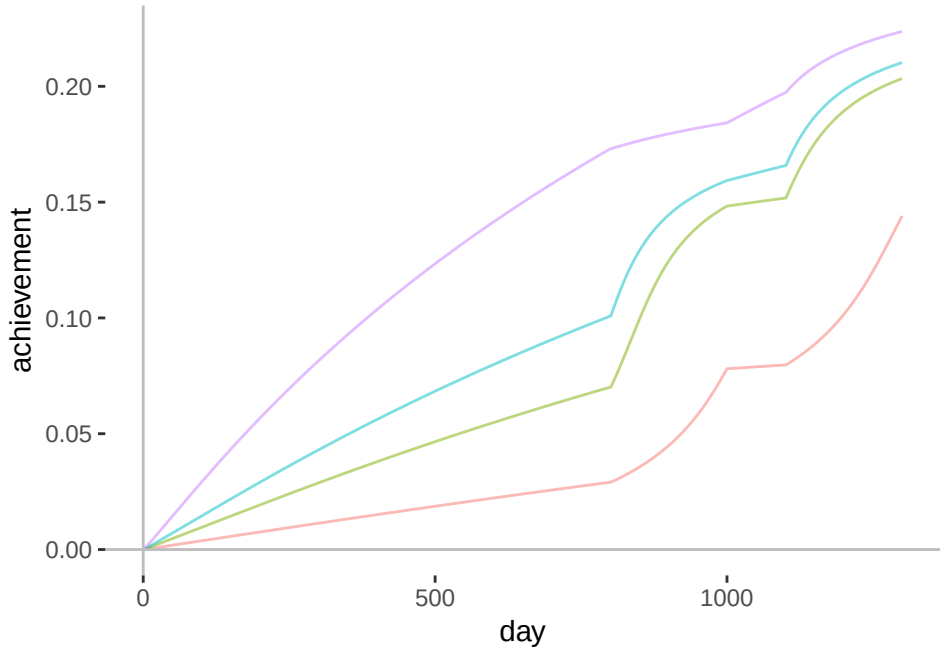
```
head(trajectories)[1:6]
```

```
##   student day1         day2         day3         day4         day5
## 1       1    0 3.815010e-05 7.630034e-05 0.0001144508 0.0001526014
## 2       2    0 9.538001e-05 1.907945e-04 0.0002862443 0.0003817301
## 3       3    0 1.430772e-04 2.862503e-04 0.0004295220 0.0005728947
## 4       4    0 2.861687e-04 5.727691e-04 0.0008598220 0.0011473485
```

The `visualizeTrajectories()` function plots the resulting output using `ggplot`. It can be used to visualize the output from the `ZPDGrowthTrajectories()` function, regardless of whether the output is pro-

duced in the `output.format="wide"` or `output.format="long"` formats. (If the output is "wide", the `visualizeTrajectories()` function internally restructures it to "long" for plotting).

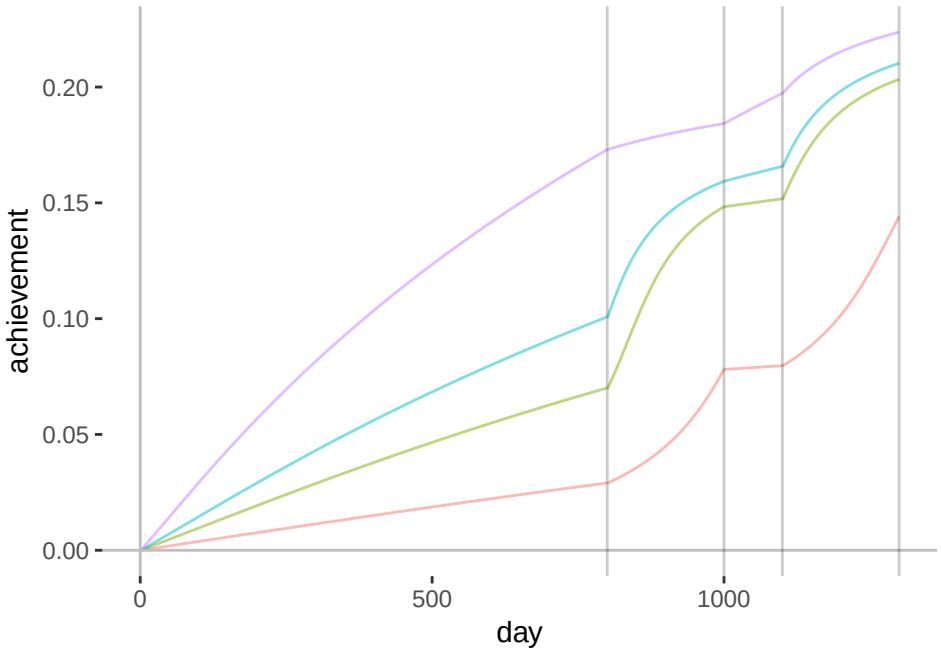
```
visualizeTrajectories(trajectories)
```



The product of the `visualizeTrajectories()` function is a `ggplot()` object and can be modified using typical `ggplot` options and functions. However, you will need to have installed the `ggplot2` package in order to modify the plot. (You can install the package, if needed, by running `install.packages("ggplot2")` at the console).

In this example the `geom_vline()` function is used to annotate the plot with vertical lines at days 800, 1000, 1100, and 1300, indicating the beginning and ending days of the Kindergarten and first grade as described by the `assignment` object created above. The `alpha` argument controls the transparency of the vertical lines.

```
figure <- visualizeTrajectories(trajectories)
figure + ggplot2::geom_vline(xintercept=c(800, 1000, 1100, 1300), alpha=.2)
```



And here's the same situation computed with the `multicore` version of the function. Note the `n.cores=4` argument.

```
trajectories.mc <- ZPDGrowthTrajectories_multicore(
  n.cores=4,
  output.format="wide",
  days=1300,
  assignment=assignment,
  curriculum.start.points=curriculum.start.points,
  curriculum.widths=curriculum.widths,
  dosage=.8,
  learning.rates=learning.rates,
  decay.rates=decay.rates,
  initial.achievements=initial.achievements,
  home.environments=home.environments,
  integration.points=200,
  curriculum.lower.slope=8,
  curriculum.upper.slope=300,
  alpha=1,
  home.curriculum.shape1=1,
  home.curriculum.shape2=5,
  zpd.offset=.03,
  zpd.sd=.028,
  zpd.scale=.15,
  decay.weight=.005,
  useGPU=FALSE,
  verbose=FALSE)
```

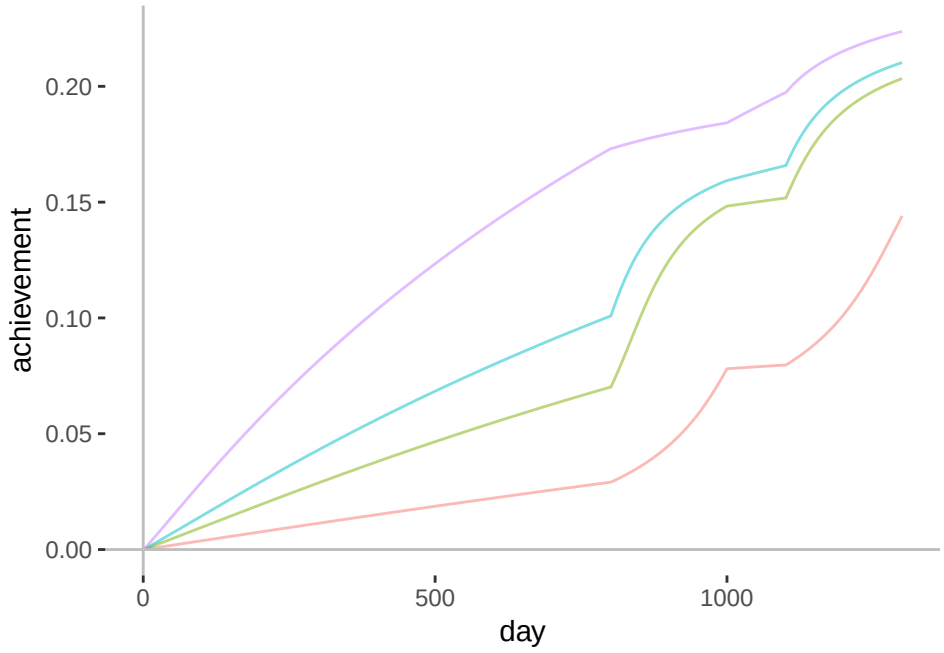
The output of the single core and multicore versions of the `ZPDGrowthTrajectories` functions are identical.

```
head(trajectories.mc)[1:6]
```

```
##   student day1         day2         day3         day4         day5
## 1       1    0 3.815010e-05 7.630034e-05 0.0001144508 0.0001526014
## 2       2    0 9.538001e-05 1.907945e-04 0.0002862443 0.0003817301
```

```
## 3       3    0 1.430772e-04 2.862503e-04 0.0004295220 0.0005728947
## 4       4    0 2.861687e-04 5.727691e-04 0.0008598220 0.0011473485
```

```
visualizeTrajectories(trajectories.mc)
```



Example

Growth rate depends on current achievement

Three students will be created that are identical in every way except for their initial achievement. The school and home curricula will be defined, and then the growth trajectories will be created and visualized

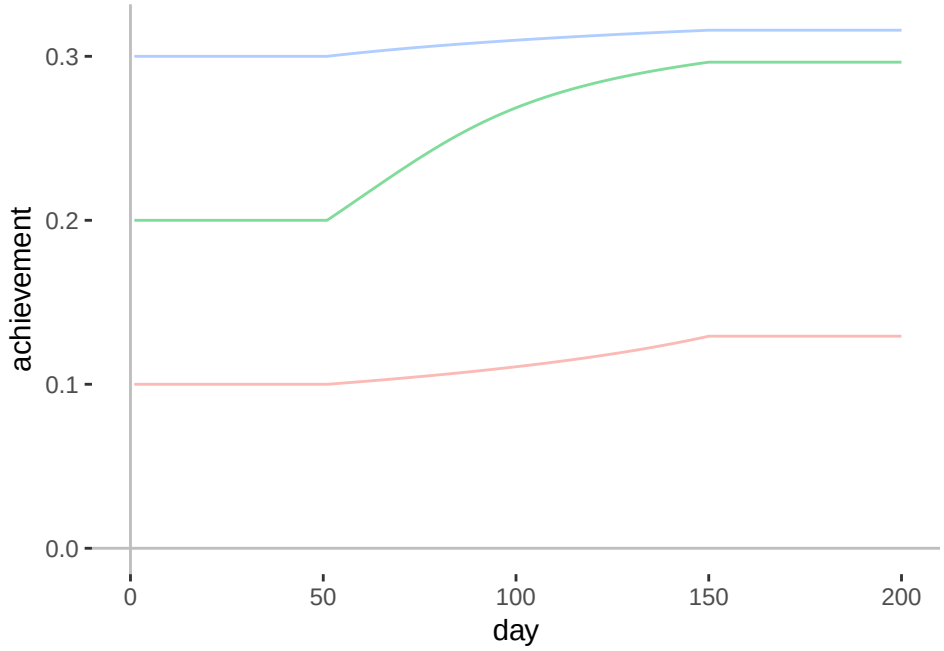
```
## *** set child characteristics *** ##
# learning rate
learning <- rep(.2, times=3)
initial <- c(.1, .2, .3)
home <- rep(0, times=3)
decay <- rep(0, times=3)


## *** set school curriculum characteristics *** ##
# assignment vector
assignment <- c(rep(0, 50), rep(1, 100), rep(0, 50))
# school curriculum
sch.curr.starts <- matrix(.2, nrow=1, ncol=1)
sch.curr.width <- matrix(.1, nrow=1, ncol=1)

## *** build the trajectories *** ##
trajectories <- ZPDGrowthTrajectories(output.format="wide", days=200,
                    assignment=assignment,
                    curriculum.start.points=sch.curr.starts,
                    curriculum.widths=sch.curr.width,
```

```
                    dosage=.8, learning.rates=learning, decay.rates=decay,
                    initial.achievements=initial, home.environments=home,
                    integration.points=200, curriculum.lower.slope=8,
                    curriculum.upper.slope=300, alpha=1, home.curriculum.shape1=1,
                    home.curriculum.shape2=5, zpd.offset=.03, zpd.sd=.028,
                    zpd.scale=.15, decay.weight=.005, useGPU=FALSE, verbose=FALSE)

## *** visualize them *** ##
visualizeTrajectories(trajectories)
```



The students who encounter a school curriculum that above the ZPD (the red line) or below the ZPD (blue line) show minimal growth during the academic year. However, the student whose ZPD is well-matched to the school curriculum exhibits rapid growth. This has nothing to do with any differences in learning rate, decay rate, or the home environment, as these are identical across students. This behavior is solely a consequence of the interaction between learning and current achievement.

The synthetic growth trajectories generated by the `ZPDGrowthTrajectories()` function have many potential uses. For example, the amount of learning (change in academic achievement) for the three simulated students can be calculated using the following code. These types of computations are facilitated by setting `output.format="wide"`.

```
# learning (change in achievement) for the three children:
trajectories$day200 - trajectories$day1
```

```
## [1] 0.02932253 0.09643906 0.01592836
```

It is also possible to fit statistical models to the synthetic data. This is useful for attempting to reproduce empirical results from synthetic data. The process is as follows:

1. Produce synthetic data.
2. Rescale achievement to the desired metric.
3. Add measurement error.
4. Extract samples ("slices") from the synthetic data at desired time points.
5. Fit statistical models.