# Using the eChem Package

*2018-06-22*

## Functions to Simulate Electrochemistry Experiments

Table 1 lists the four functions in this package that simulate electrochemistry experiments. Each function takes a set of arguments that define the experiment and each returns an extensive set of results as a list.

Table 1: Functions to Simulate Electrochemistry Experiments

| function | electrochemistry experiment |
|---|---|
| `simulateCA` | chronoamperometry (single and double pulse) |
| `simulateCC` | chronocoulometry (single and double pulse) |
| `simulateCV` | cyclic voltammetry |
| `simulateLSV` | linear-sweep voltammetry (with and without stirring) |

### Simulating a Cyclic Voltammetry Experiment

The format for the function `simulateCV` is shown here

```
simulateCV = function(e.start = 0.0, e.switch = -0.5, e.form = -0.25,
                      mechanism = c("E", "EC", "CE"),
                      ko = 1, kcf = 0, kcr = 0,
                      n = 1, alpha = 0.50, d = 1e-5, area = 0.01,
                      temp = 298.15, scan.rate = 1.0, conc.bulk = 1e-3,
                      t.units = 2000, x.units = 180, sd.noise = 0)
```

with its arguments and their default values defined below (units shown in parentheses):

- `e.start`: initial potential (V); defaults to 0.0 V
- `e.switch`: swiching potential (V); defaults to –0.5 V
- `e.form`: standard state formal potential for the redox couple (V); defaults to –0.25 V
- `mechanism`: options for redox reaction only, `E`, a redox reaction with a following chemical reaction, `EC`, or a redox reaction with a preceding chemical reaction, `CE`; defaults to `E`
- `ko`: standard heterogeneous electron transfer rate constant (cm/s); defaults to 1.0 cm/s
- `kcf`, `kcr`: homogeneous rate constants for the forward and the reverse chemical reaction ($s^{-1}$); defaults to 0 as the default mechanism is redox-only
- `n`: electrons for redox couple (unitless); defaults to 1
- `alpha`: transfer coefficient (unitless); defaults to 0.5
- `d`: diffusion coefficient for Ox and Red ($cm^2/s$); defaults to $1.0 \times 10^{-5}$ $cm^2/s$
- `area`: surface area of the electrode ($cm^2$); defaults to 0.010 $cm^2$
- `temp`: temperature in Kelvin; defaults to 298.15 K
- `scan.rate`: rate at which the potential is changed (V/s); defaults to 1.0 V/s
- `conc.bulk`: total concentration of Ox, Red, and Z (mol/L); defaults to $1.0 \times 10^{-3}$ mol/L
- `t.units`: number of time units in diffusion grid (unitless); defaults to 2000
- `x.units`: number of distance units in diffusion grid (unitless); defaults to 180
- `sd.noise`: standard deviation for noise as percent of maximum current (µA); defaults to 0

To use the `simulateCV` function, we assign it to an object and pass along values for the function's arguments. For example, if we wish to accept the function's default values—as defined above—then we enter the following line of code where `cv1` is the object that contains the simulation's results.

```
cv1 = simulateCV()
```

If we wish to change the value for any of the function's arguments, then we simply enter those values within the parentheses; thus, to simulate a two-electron reduction and a scan rate of 0.1 V/s—and here we also change the number of time units and the number of distance units to decrease the size of the resulting file—we enter the following line of code

```
cv2 = simulateCV(n = 2, scan.rate = 0.1, t.units = 1000, x.units = 100)
```

Once created, we can use R's structure command, `str()`, to examine the information stored within the object as a list:

```
str(cv2)
```

```
## List of 31
##  $ expt       : chr "CV"
##  $ mechanism  : chr "E"
##  $ file_type  : chr "full"
##  $ current    : num [1:1001] 0.00 1.23e-07 9.87e-08 9.15e-08 8.95e-08 ...
##  $ potential  : num [1:1001] 0 -0.001 -0.002 -0.003 -0.004 -0.005 -0.006 -0.007 -0.008 -0.009 ...
##  $ time       : num [1:1001] 0 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 ...
##  $ distance   : num [1:101] 0 0.0006 0.0012 0.0018 0.0024 0.003 0.0036 0.0042 0.0048 0.0054 ...
##  $ oxdata     : num [1:1001, 1:101] 1 1 1 1 1 ...
##  $ reddata    : num [1:1001, 1:101] 0.00 3.82e-09 4.13e-09 4.46e-09 4.83e-09 ...
##  $ chemdata   : num [1:1001, 1:101] 0 0 0 0 0 0 0 0 0 0 ...
##  $ formalE    : num -0.25
##  $ initialE   : num 0
##  $ switchE    : num -0.5
##  $ electrons  : num 2
##  $ ko         : num 1
##  $ kcf        : num 0
##  $ kcr        : num 0
##  $ alpha      : num 0.5
##  $ diffcoef   : num 1e-05
##  $ area       : num 0.01
##  $ temperature: num 298
##  $ scanrate   : num 0.1
##  $ conc.bulk  : num 0.001
##  $ tunits     : num 1000
##  $ xunits     : num 100
##  $ sdnoise    : num 0
##  $ direction  : num -1
##  $ k_f        : num [1:1001] 5.95e-05 6.18e-05 6.43e-05 6.68e-05 6.95e-05 ...
##  $ k_b        : num [1:1001] 16820 16178 15561 14967 14395 ...
##  $ jox        : num [1:1001] 0.00 -6.37e-17 -5.11e-17 -4.74e-17 -4.64e-17 ...
##  $ jred       : num [1:1001] 0.00 6.37e-17 5.11e-17 4.74e-17 4.64e-17 ...
```

In addition to returning the function's many inputs, the list also includes scalers that return the direction in which the potential was scanned (−1 for a cathodic scan or +1 for a anodic scan) and the type of file (full data or reduced data; more on this later), vectors that return the times and distances, the current as a function of time, the potential as a function of time, the forward and the reverse heterogenous electron transfer rate constants as a function of time, and the flux of Ox and Red as a function of time, and matrices that return values for the concentrations of Ox, Red, and Z as a function of time and of distance. Items in this list are used by other functions that display the result of the simulated cyclic voltrammetry experiment.

## Simulating a Linear Sweep Voltammetry Experiment

The format for `simulateLSV` is shown here

```
simulateLSV = function(e.start = 0.0, e.end = -1, e.form = -0.25,
                  mechanism = c("E", "EC", "CE"),
                  ko = 1, kcf = 0, kcr = 0,
                  n = 1, alpha = 0.50, d = 1e-5, area = 0.01,
                  temp = 298.15, scan.rate = 1.0, conc.bulk = 1e-3,
                  t.units = 2000, x.units = 180, sd.noise = 0,
                  stir.rate = c("off", "slow", "medium", "fast"))
```

With two exceptions, the arguments passed to `simulateLSV` are identical to those included in `simulateCV`; the two unique arguments are

- `e.end`: final potential (V); defaults to -1.0 V
- `stir.rate`: rate at which the solution is stirred (unitless); one of `off`, `slow`, `medium`, or `fast`, with a default of `off`

The function is used in the same way as with `simulateCV`, returning the results as a list; thus, to simulate a linear sweep voltammogram using all default conditions but with maximum stirring, we enter the following `R` code

```r
lsv1 = simulateLSV(stir.rate = "fast")
```

## Simulating a Chronoamperometry Experiment

The format for `simulateCA` is shown here

```
simulateCA = function(e.start = 0.0, e.pulse = -0.5, e.form = -0.25,
                      mechanism = c("E", "EC", "CE"),
                      ko = 1, kcf = 0, kcr = 0,
                      pulses = c("single", "double"),
                      t.1 = 10, t.2 = 0, t.end = 30,
                      n = 1, alpha = 0.50, d = 1e-5, area = 0.01,
                      temp = 298.15, conc.bulk = 1e-3,
                      t.units = 2000, x.units = 180, sd.noise = 0)
```

Many of the arguments passed to `simulateCA` are identical to those included in `simulateCV`; arguments unique to `simulateCA` are

- `e.pulse`: potential at the end of the first pulse (V); defaults to –0.5 V
- `pulses`: either `single` or `double` with the first pulse from `e.start` to `e.pulse` and the second pulse, when included, returning the potential to `e.start`; defaults to a single pulse experiment
- `t.1`: time at which the first pulse is applied (s); defaults to 10 s
- `t.2`: time at which the second pulse is applied (s); defaults to 0 s for a single pulse experiment
- `t.end`: time at which experiment ends (s); defaults to 30 s

The function is used in the same way as with `simulateCV`, returning the results as a list; thus, to simulate a double step chonoamperometry experiment with potential pulses at 5 s and 10 s, a total time of 15 s—and all other conditions set to their default values—we enter the following `R` code

```r
ca1 = simulateCA(pulses = "double", t.1 = 5, t.2 = 10, t.end = 15)
```

## Simulating a Chronocoulometry Experiment

The format for `ccSim`, which is shown here

```
simulateCC = function(filename)
```

takes as its only argument the name of an object created using `simulateCA`. The function completes a trapezoidal integration to convert the chronoamperogram (current vs. time) into a chronocoulomgram (charge vs. time) and returns to its assigned object a list of results; thus, to simulate the chronocoulogram based on the chronoamperometry experiment described above, we enter the following `R` code

```
cc1 = simulateCC(ca1)
```

## Functions to Reduce the Size of a Simulation's File

The four simulation functions described above create moderately large data files because they return data for all aspects of the simulation's computation. For example, the file returned when running a simulation using the default condtions `simulateCV` is 2.4 Mb in size. The four `sample` functions

Table 2: Functions to Sample a Simulation

| function | operates on object created using... |
|---|---|
| `sampleCA` | `simulateCA` |
| `sampleCC` | `simulateCC` |
| `sampleCV` | `simulateCV` |
| `sampleLSV` | `simulateLSV` |

return a limited amount of data: the type of experiment, an indication that the data is `reduced`, and vectors of currents (`sampleCA`, `sampleCV`, and `sampleLSV`) or charges (`sampleCC`), and vectors of potentials (`sampleCV` and `sampleLSV`) or times (`sampleCA` and `sampleCC`). All four functions have the same general format, as shown here for `sampleCV`

```
sampleCV = function(filename, data.reduction = 1)
```

where `filename` is the name of the object that contains results for the original simulation, and `data.reduction` is the percent of data to retain (which is evenly spaced over time); thus, the following `R` code returns 10% of the data from a simulated cyclic voltammetry experiment.

```
cv2_sample = sampleCV(cv2, data.reduction = 10)
str(cv2_sample)
```

```
## List of 4
##  $ expt     : chr "CV"
##  $ file_type: chr "reduced"
##  $ current  : num [1:101] 0.00 1.13e-07 2.28e-07 4.89e-07 1.06e-06 ...
##  $ potential: num [1:101] 0 -0.01 -0.02 -0.03 -0.04 -0.05 -0.06 -0.07 -0.08 -0.09 ...
```

## Functions to Examine the Result of a Simulation

As described in the previous sections, the functions in Table 1 and Table 2 return a list of values that include parameters passed to the functions and the results of the simulation itself. In this section we consider how to examine the results of a simulation either as a static visualization or as a dynamic visualization.

## Static Visualizations

There are two basic types of static visualizations: `plot`, which provides a variety of options for visualizing the results of one or, in some cases, more electrochemical simulations

Table 3: Static Visualizations Using the `plot` Functions

| function | operates on object created by... | displays... |
|---|---|---|
| `plotCA` | `simulateCA`, `sampleCA` | current vs. time for 1–5 simulations |
| `plotCC` | `simulateCC`, `sampleCC` | charge vs. time for 1–5 simulations |
| `plotCV` | `simulateCV`, `sampleCV` | current vs. potential for 1–5 simulations |
| `plotLSV` | `simulateLSV`, `sampleLSV` | current vs. potential for 1–5 simulations |
| `plotPotential` | `simulateCA`, `simulateCC`, `simulateCV`, `simulateLSV` | potential vs. time |
| `plotDiffusion` | `simulateCA`, `simulateCC`, `simulateCV`, `simulateLSV` | concentrations of Ox, Red, and Z vs. distance at a specified time |
| `plotGrid` | `simulateCA`, `simulateCC`, `simulateCV`, `simulateLSV` | diffusion profiles for Ox, Red, and Z at eight equally-spaced times, arranged around a central plot that shows the chronoamperogram, chronocoulogram, cyclic voltammogram, or linear sweep voltammogram |
| `plotDiffGrid` | `simulateCA`, `simulateCC`, `simulateCV`, `simulateLSV` | concentration of Ox, Red, Z, or all three as a function of time and distance |

and `annotate`, which displays the result for a single simulation along with the characteristic values used to evaluate electrochemical and chemical reversibility.

Table 4: Static Visualizations Using the `annotate` Functions

| function | operates on object created by... | displays... |
|---|---|---|
| `annotateCA` | `simulateCA` | current vs. time annotated with characteristic values for the current at a specified time following each pulse and, for a double-pulse experiment, the resulting current ratio |
| `annotateCC` | `simulateCC` | charge vs. time annotated with characteristic values for the charge at a specified time following each pulse and, for a double-pulse experiment, the resulting charge ratio |
| `annotateCV` | `simulateCV` | current vs. potential annotated with characteristic values for the peak potentials, the peak currents, the average and the difference in the peak potentials, and the peak current ratio |

| function | operates on object created by… | displays… |
|---|---|---|
| `annotateLSV` | `simulateLSV` | current vs. potential annotated with characteristic values for the peak potential and the peak current |

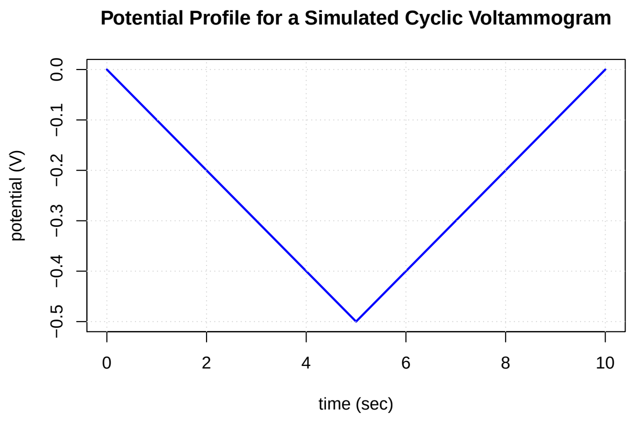
## Static Visualizations for a Cyclic Voltammetry Simulation

To view the potential profile for a simulation, we use the `plotPotential` function, which takes the following form

```
plotPotential = function(filename, main_title = NULL)
```

where `filename` is the name of an object that contains the results of a simulation and `main_title` is an optional title supplied as a character string; thus

```r
plotPotential(cv2, main_title = "Potential Profile for a Simulated Cyclic Voltammogram")
```

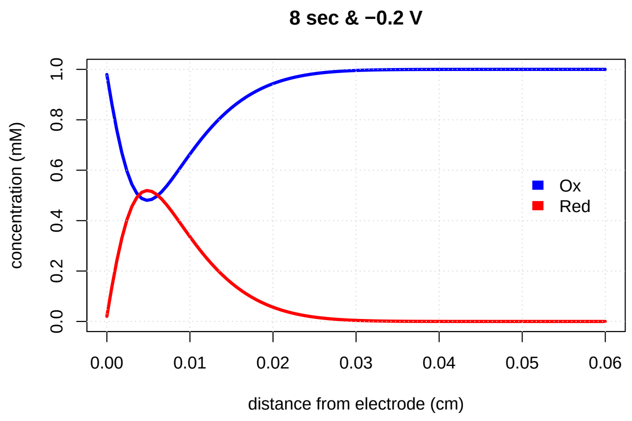


To examine the diffusion profile for a simulation, we choose from among three options:

```
plotDiffusion = function(filename, t = 1)
plotGrid = function(filename)
plotDiffGrid = function(filename, species = c(TRUE, TRUE, FALSE),
                         scale.factor = 1)
```

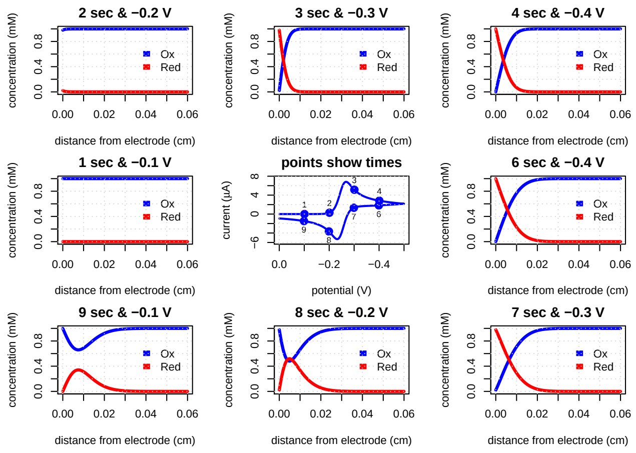
where `filename` is the name of an object that contains the results of a simulation, `t` is the selected time, `species` indicates whether the diffusion profile is returned for, in order, Ox, Red, and Z, and `scale.factor` allows for adjusting the distances being displayed. The function `plotDiffusion` returns the diffusion profiles for Ox, Red, and Z (where appropriate) as a function of distance from the electrode at the specified time.

```r
plotDiffusion(cv2, t = 8)
```



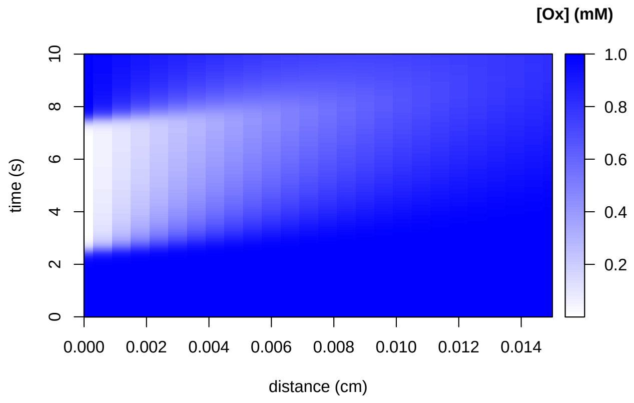
The function `plotGrid` returns a 3 × 3 grid that consists of eight diffusion profiles—equally spaced in time—arranged around a central plot of the simulation's cyclic voltammogram.

```r
plotGrid(cv2)
```



The function `plotDiffGrid` returns a heatmap showing the concentrations of Ox, Red, and/or Z as a function of time and distance; here we choose to display only Ox and to display only the first 25% of the distance from the electrode surface

```r
plotDiffGrid(cv2, species = c(TRUE, FALSE, FALSE), scale.factor = 0.25)
```

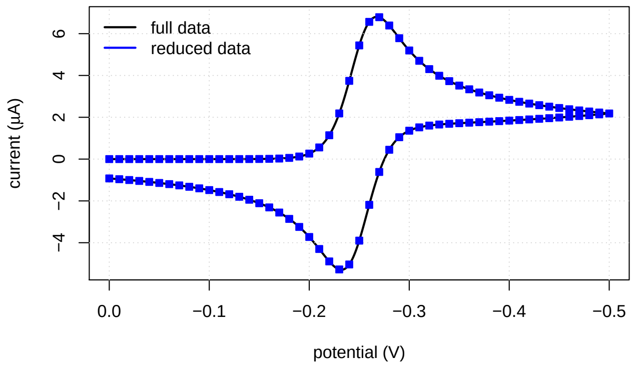


The function `plotCV` is used to display the cyclic voltammogram for 1–5 simulations, making it easy to examine how a change in simulation parameters affects the cyclic voltammogram. The function's format is shown here

```
plotCV = function(filenames = list(file1 = NULL, file2 = NULL),
                        legend_text = NULL,
                        legend_position = c("topleft", "topright",
                                            "bottomleft", "bottomright"),
                        main_title = NULL,
                        line_widths = c(2, 2, 2, 2, 2),
                        line_types = c(1, 2, 3, 4, 5),
                        point_symbols = c(21, 22, 23, 24, 25),
                        line_colors = c("blue", "blue", "blue",
                                        "blue", "blue"))
```

where `filenames` is a list that contains 1–5 objects created using `simulateCV` or `sampleCV`, `legend_text` is an optional vector with descriptive character strings for each simulation, `legend_position` is a character string that indicates where the legend is placed if `legend_text` is not NULL, `main_title` is an optional character string, and `line_widths`, `line_types`, and `line_colors` specify the width, type, and color of the lines used for each cyclic voltammogram. The code below, for example, plots the full and reduced cyclic voltammograms that we created earlier and adds a legend.

```
plotCV(filenames = list(cv2, cv2_sample),
       legend_text = c("full data", "reduced data"),
       line_colors = c("black", "blue"),
       line_types = c(1, 1))
```



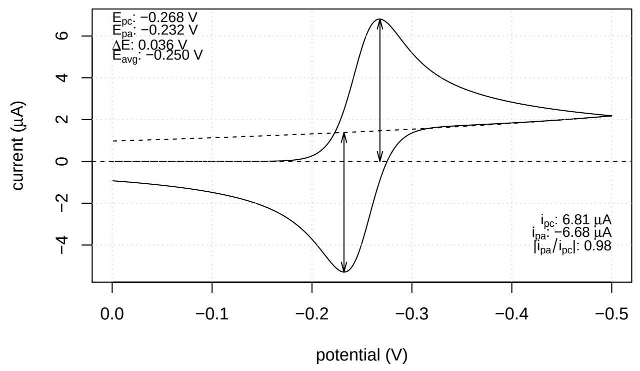
The line types are 1 = solid, 2 = dashed, 3 = dotted, 4 = dot dash, and 5 = long dash, and the point symbols are circle = 21, square = 22, diamond = 23, upward triangle = 24, and downward triangle = 25. As is evident from this example, it is not necessary to match the number of values in `line_widths`, `line_types`, and `line_colors` to the number of files passed to the function.

Finally, the `annotateCV` function returns a plot of the cyclic voltammogram that is annotated with key characteristic values used to evaluate electrochemical and chemical reversibility: the cathodic peak potential, $E_{p,c}$, the anodic peak potential, $E_{p,a}$, the difference between the peak potentials, $\Delta E$, the average peak potential, $E_{\text{avg}}$, the cathodic peak current, $i_{p,c}$, the anodic peak current, $i_{p,a}$, and the peak current ratio, $|i_{p,c}|/|i_{p,a}|$. The function's format is shown here

```
annotateCV = function(filename, forward.per = 5, reverse.per = 5,
                      threshold = 0.05)
```

where `filename` is a single object that contains the results of a simulated cyclic voltammogram. The arguments `forward.per` and `reverse.per` define the percentage of data used to determine the peak currents for the forward and the reverse scans: for the forward scan the default is the first 5% of the data and for the reverse scan the default is the first 5% of the data following the switching potential. The `threshold` argument is the minimum measurable current to report, in µA.

```
annotateCV(cv2)
```



See the document `Additional Examples` for other ideas on how to create static visualizations for chronomperometry, chronocoulometry, cyclic voltammetry, and linear sweep voltammetry experiments.

## Dynamic Visualizations

A static visualization provides a single fixed view of a simulation; a dynamic visualization displays the simulation as an animation that consists of 40 frames. As outlined in Table 5, each of the `animate` functions shows both how the diffusion profiles for Ox, Red, and Z (when using an `EC` or a `CE` mechanism) develop over time as the chronoamperogram, the chronocoulogram, the cyclic voltammogram, or the linear sweep voltammogram develops.

Table 5: Dynamic Visualizations Using the `animate` Functions

| function | operates on object created by… | shows… |
|---|---|---|
| `animateCA` | `simulateCA` | how the diffusion profiles for Ox, Red, and (where appropriate) Z changes as a function of time and how the current changes as a function of time |
| `animateCC` | `simulateCC` | how the diffusion profiles for Ox, Red, and (where appropriate) Z changes as a function of time and how the charge changes as a function of time |
| `animateCV` | `simulateCV` | how the diffusion profiles for Ox, Red, and (where appropriate) Z changes as a function of time and how the current changes as a function of potential |
| `animateLSV` | `simulateLSV` | how the diffusion profiles for Ox, Red, and (where appropriate) Z changes as a function of time and how the current changes as a function of potential |

## Dynamic Visualization for a Cyclic Voltammetry Simulation

The `animateCV` function takes the following form

```
animateCV = function(filename, out_type = c("html", "gif"), out_name = "aniCV")
```

where `filename` is the name of the object created using `simulateCV`, `out_type` gives the type of animation, which is either an `html` (default) or a `gif` animation, and `out_name` is used to construct names for the files the function creates. For example, the following code

```r
animateCV(cv1, out_type = "gif", out_name = "cv1")
```

which is not executed here, creates a single file with name `cv1.gif`, which contains the images that make up the animation. When using the option to create an `html` animation, the function creates folders with the necessary CSS styling and javascript, a folder with the name `cv1_dir` that contains image files that make up the animation, and an html file with the name `cv1.html` with the html code to display the animation in a browser. The other `animate` functions work in the same way. *Note that all files created by the `animate` functions are saved to the current working directory.*