How Big is Your Graph?
An R Cheat Sheet

Introduction
All functions that open a device for graphics will have **height** and **width** arguments to control the size of the graph and a **pointsize** argument to control the relative font size. In **knitr**, you control the size of the graph with the chunk options, **fig.width** and **fig.height**. This sheet will help you with calculating the size of the graph and various parts of the graph within R.

**Your graphics device**
- `dev.size()` (width, height)
- `par("din")` (r.o.) (width, height) in inches

Both the `dev.size` function and the `din` argument of `par` will tell you the size of the graphics device. The `dev.size` function will report the size in
- 1. inches (`units="in"`), the default
- 2. centimeters (`units="cm"`)
- 3. pixels (`units="px"`)

Like several other `par` arguments, `din` is read only (r.o.) meaning that you can ask its current value (`par("din")`) but you cannot change it (`par(din=c(5,7))` will fail).

**Your plot margins**
- `par("mar")` (bottom, left, top, right) in inches
- `par("mar")` (bottom, left, top, right) in lines

Margins provide you space for your axes, axis, labels, and titles. A 'line' is the amount of vertical space needed for a line of text.

If your graph has no axes or titles, you can remove the margins (and maximize the plotting region) with
- `par(mar=rep(0,4))`

**Getting a square graph**
- `par("pty")`

You can produce a square graph manually by setting the width and height to the same value and setting the margins so that the sum of the top and bottom margins equal the sum of the left and right margins. But a much easier way is to specify `pty="s"`, which adjusts the margins so that the size of the plotting region is always square, even if you resize the graphics window.

**Converting units**
For many applications, you need to be able to translate user coordinates to pixels or inches. There are some cryptic shortcuts, but the simplest way is to get the range in user coordinates and measure the proportion of the graphics device devoted to the plotting region.

- `user.range <- par("usr")[c(2,4)] - par("usr")[c(1,3)]`
- `region.pct <- par("plt")[c(2,4)] - par("plt")[c(1,3)]`
- `region.px <- dev.size(units="px") * region.pct`
- `px.per.xy <- region.px / region.pct`
- `px.per.xy1 <- user.range`

To convert a horizontal or distance from the x-coordinate value to pixels, multiply by `px.per.xy1[1]`. To convert a vertical distance, multiply by `region.px.per.xy[2]`. To convert a diagonal distance, you need to invoke Pythagoras.

- `a.px <- x.dist*px.per.xy[1]`
- `b.px <- y.dist*px.per.xy[2]`
- `c.px <- sqrt(a.px^2+b.px^2)`

To rotate a string to match the slope of a line segment, you need to convert the distances to pixels, calculate the arc tangent, and convert from radians to degrees.

- `segments(x0, y0, x1, y1)`
- `delta.x <- (x1 - x0) * px.per.xy[1]`
- `delta.y <- (y1 - y0) * px.per.xy[2]`
- `angle.radians <- atan2(delta.y, delta.x)`
- `angle.degrees <- angle.radians * 180 / pi`
- `text(x1, y1, "TEXT", srt=angle.degrees)`

---

CC BY Steve Simon, P.Mean Consulting • mail@pmean.com • https://www.rstudio.com/resources/cheatsheets/
If you display multiple plots within a single graphics window (e.g., with the `mfrow` or `mfcol` arguments of `par` or with the `layout` function), then the `fig` and `fin` arguments will tell you the size of the current subplot window in percent or inches, respectively.

Each subplot will have margins specified by `mai` or `mar`, but no outer margin around the entire set of plots, unless you specify them using `oma`, `omd`, or `omi`. You can place text in the outer margins using the `mtext` function with the argument `outer=TRUE`.

The `mfg` argument of `par` will allow you to jump to a subplot in a particular row and column. If you query with `par("mfg")`, you will get the current row and column followed by the maximum row and column.

### Character and string sizes

The `strheight()` function will tell you the height of a specified string in inches.

For a single line of text, `strheight()` will give you the height of the letter "M". If you have a string with one or more linebreaks ("\n"), the `strheight()` function will measure the height of the letter "M" plus the height of one or more additional lines. The height of a line is dependent on the line spacing, set by the `height` argument of `par`. The default line height (`heights=1`), corresponding to single spaced lines, produces a line height roughly 1.5 times the height of "M".

The `strwidth()` function will produce different widths to individual characters, representing the proportional spacing used by most fonts (a "W" using much more space than an "i"). For the width of a string, the `strwidth()` function will sum up the lengths of the individual characters in the string.

### If your fonts are too big or too small

Fixing this takes a bit of trial and error.

1. Specify a larger/smaller value for the `pointsize` argument when you open your graphics device.
2. Try opening your graphics device with different values for `height` and `width`. Fonts that look too big might be better proportioned in a larger graphics window.
3. Use the `cex` argument to increase or decrease the relative size of your fonts.

### If your axes don't fit

There are several possible solutions.

1. You can assign wider margins using the `mar` or `mai` argument in `par`.
2. You can change the orientation of the axis labels with `las`. Choose among
   a. `las=0` both axis labels parallel
   b. `las=1` both axis labels horizontal
   c. `las=2` both axis labels perpendicular
   d. `las=3` both axis labels vertical.
3. Change the relative size of the font
   a. `cex.axis` for the tick mark labels.
   b. `cex.lab` for `xlab` and `ylab`.
   c. `cex.main` for the main title
   d. `cex.sub` for the subtitle.