Applied Mathematical Economics with *Mathematica*

Author: Alan G. Isaac
Institution: American University

Initialization

Front Matter

Preface

This booklet is intended to support the use of *Mathematica* in courses on Mathematical Economics. No prior programming experience is assumed; no prior *Mathematica* experience is assumed. Built-in *Mathematica* commands are introduced more-or-less on an as-needed basis, rather than systematically. Some commands are treated in detail, but I make no effort to reproduce the excellent and extensive *Mathematica* documentation. Instead, I introduce readers to the *Mathematica* help system, and I occasionally point to online resources.

The mathematics presented herein is not self-contained. This is not a textbook in Mathematical Economics. Rather, I use *Mathematica* to provide illustrations of some mathematical tools typically covered in a first course in Mathematical Economics. Core goals include the following:

- introduce the basic use of *Mathematica* notebooks for computation, plotting, and symbolic algebra
- encourage readers to improve their mathematical intuition with concrete applications and graphical explorations
- familarize readers with the computational application of basic mathematical tools to economic problems

This booklet approaches these goals by providing economics-relevant applications of the following:

- linear algebra
- univariate and multivariate calculus
- linear and nonlinear comparative statics
- univariate and mulitvariate optimization

Use of *Mathematica* provides substantial benefits to teachers and students in introductory Mathematical Economics. Courses introducing mathematical tools to economists often cover a wide range of topics at a speed that can intimidate students. Concrete applications of the mathematical concepts help students learn, but these often require tedious calculations that consume classroom time as well as the mental energy of students. The use of *Mathematica* delegates some of this tedium to the software and

greatly increases the speed with which examples can be developed and problems solved. This allows greater emphasis on core mathematical practices.

This booklet was written using *Mathematica* version 10 for Windows. Users on other operating systems should have no difficulties as long as they have a recent version of *Mathematica*. However, I use a few commands that did not exist in earlier versions, and sometimes of the algorithms associated with a command improve in later versions. Therefore readers are encouraged to work the examples with version 10 or later.

Some sections or passages are marked as “advanced”. This material can be skipped on a first reading by those new to *Mathematica*, but those with some *Mathematica* exposure should find them accessible even on a first reading.

Mathematica and the Notebook: A Brief Introduction

This chapter provides a basic introduction to the use of *Mathematica* notebooks to enter text and to perform basic numerical and symbolic computations.

Notebooks and the Front End

Mathematica comes with an interactive graphical user interface (GUI) called the *front end*. We use the front end to perform interactive computations and to keep a record of these computations as notebooks. To create a new notebook, pick from the *front end* menus File > New > Notebook.

Evaluating Numerical Expressions

Cells

A *Mathematica* notebook is a collection of cells. After starting the *Mathematica* front end and creating a new notebook, just start typing to create a notebook cell. You will see that the cell is delimited by a bracket on its right side. Press the down arrow to leave this cell. A horizontal line will appear, indicating an insertion point for a new cell. Type some more and a new cell is created.

Cell Styles

Click on a cell’s right edge to make it active, and then use the Format > Style menu to give it a style. The default cell style is Input, which is for *Mathematica* expressions that you wish to evaluate, but you can change the style at any time by using the Format menu. We will most often be creating one of the following Cell types: Input, DisplayFormula, and Text. We will also use Title, Section, and Subsection.

There are also keyboard shortcuts for the Format menu. (See https://reference.wolfram.com/language/tutorial/KeyboardShortcutListing.html#1484044289).

Text cells can contain ordinary text. Give a cell a Text style when you want to provide discussion or

descriptions of your calculations. The *Mathematica* front end will do some basic formatting of this text, in a manner similar to how a word processor handles text.

In contrast, use the default Input style if your cell contains *Mathematica* expressions that you want to evaluate.

If you want to create a new cell with the same style your are currently using, press Alt+Enter.

Evaluating Input Cells

After entering any *Mathematica* expression into an Input cell, press `Shift+Enter` to evaluate it. The result is placed in its own cell, which has the Output style. For example, consider an arbitrary arithmetic expression.

```
105 / (15 * 7)
```

```
1
```

This example shows that you can easily use *Mathematica* as a calculator. Note the familiar use of parentheses to determine the order of evaluation in the Input expression. (Q: What is the result if you remove them?)

We often want to explain an expression. We can do this with comments, which open with an opening parenthesis and asterisk and close with an asterisk and closing parenthesis, `(* like this *)`.

```
355 / 113.  (* crude approximation to π *)
```

```
3.14159
```

You may place more than one expression in a single Input cell. If you want to compute the value of an expression but not display it, end the expression with a semicolon. If you want to refer to the last computed value, use a percent sign. As an example of both, let us do a computation on multiple lines but only display the final result.

```
2 * 3;
% + 5
```

```
11
```

Mathematica allows you to add styles to your results. However, keep in mind that a formatted object does not behave the same as the underlying object. Once a number is styled, for example, you will not be able simply to do arithmetic with the result.

```
Style[2 * 3, Red]
% + 5 (* addition will not be performed *)
```

```
6
```

```
5 + 6
```

Numbers

Mathematica distinguishes between exact numbers (roughly, integers and rationals) and approximate numbers (roughly, floating point numbers). Computations done with exact numbers return exact results.

```
5 * (3 / 7)
```

$$\frac{15}{7}$$

We can use the `N` command to turn an exact number into an approximate number.

```
N[%]
```

```
2.14286
```

Mathematica names a few special numbers. (Like Mathematica commands, these names all start with a capital letter.)

```
{Pi, E, I, Infinity, -Infinity}
```

```
{π, e, i, ∞, -∞}
```

Use `N` to turn these into approximate numbers (when possible):

```
N[%]
```

```
{3.14159, 2.71828, 0. + 1. i, ∞, -∞}
```

Numerical Comparison with Relational Operators

Mathematica has the usual collection of comparison operators: `<`, `≤`, `==`, `≠`, `≥`, `>`. The value of a numerical comparison is "boolean" (i.e., either `True` or `False`). E.g.,

```
1 < 1
```

```
False
```

Expressions are evaluated before they are compared, and numbers of different types compare in a natural fashion.

```
1 == 1.0
1 == 3 / 3
```

```
True
```

```
True
```

We can use the `And` and `Or` commands to determine whether all or any comparisions are `True`.

```
And[1 < 1, 1 ≤ 1, 1 == 1, 1 ≠ 1, 1 ≥ 1, 1 > 1]
Or[1 < 1, 1 ≤ 1, 1 == 1, 1 ≠ 1, 1 ≥ 1, 1 > 1]
```

```
False
```

```
True
```

Each comparison operator is shorthand for a *Mathematica* command. For example, the infixed double equal sign `==` is shorthand for `Equal`.

Relational operators can be chained:

```
1 == 1 + 0 == 0 + 1
3 > 2 > 1
```

```
True
```

```
True
```

Advanced: Mathematica numerical equality are “clever”, relying on numerical approximation to test inequality. Thus for example the following comparision evaluates to `True` in Mathematica, whereas (due to rounding error) it would not in most languages.

```
.3 == .1 + .2
```

```
True
```

Advanced: If you really want to test for sameness (after evaluation) and not just numerical equality, use three equals signs: ===. For *Mathematica*, exact numbers are not the same thing as approximate numbers.

```
1 === 3 / 3
.3 === .1 + .2
1 === 1.0
```

```
True
```

```
True
```

```
False
```

Stopping an Evaluation

Do not be afraid to evaluate expressions in *Mathematica*. If you make a mistake that results in an invalid expression, *Mathematica* will tell you about your mistake. If you evaluate an expression that is taking too much time, you can pick Evaluation > Abort Evaluation from the Mathematic menus. (You can alternatively enter Alt+. to abort an evaluation, i.e., bring it to a full stop.) If you want to evaluate an expression that you fear may take a very long time, you can limit the amount of time *Mathematica* will work on it with the TimeConstrained command.

It is rare but possible to enter an expression that causes *Mathematica* to lock up, so be sure to save often. It is possible to ask *Mathematica* to autosave your work, but since half-written code may be useless or worse, you may find it more useful to save often by hand (ctrl+s).

Full Form

Each *Mathematica* expression has an equivalent “full form”, which is what the front end actually sends to the kernel for evaluation. For example, our expression `105/(15*7)` is really a convenient shorthand for **Times[105,Power[Times[15,7],-1]]**. We will seldom have reason to consider this full form, but at times it is useful for debugging. You can use the `Hold` and `FullForm` commands to discover it. (The `Hold` command prevents evaluation of the expression, and `FullForm` returns its full form.)

```
HoldForm[FullForm[105 / (15 * 7)]]
```

```
Times[105, Power[Times[15, 7], -1]]
```

```
HoldForm[FullForm[{1 < 1, 1 ≤ 1, 1 == 1, 1 ≥ 1, 1 > 1}]]
```

```
List[Less[1, 1], LessEqual[1, 1], Equal[1, 1], GreaterEqual[1, 1], Greater[1, 1]]
```

Symbols

Rules for the Creation of Symbols

We will often want to give names to *Mathematica* objects, using user-defined symbols. You can use almost any combination of letters to define a symbol. By convention, *Mathematica*'s system-defined symbols capitalize the first letter, so in order to avoid name clashes you should not capitalize the first letter of your new symbols. A symbol cannot start with a digit, but you can use numeric characters anywhere else in your symbols.

Sometimes we would like to include a non-alphanumeric character in order to visually break a symbol into parts. The standard choice in many languages is the underscore, but that has a special meaning in *Mathematica*. (It is used to produce patterns.) The only choice from a standard American keyboard is the back tick. (This actually creates a symbol with a "context", as we will discuss shortly.) Be sure not to end your symbol with a dollar sign nor with a dollar sign followed by digits, as *Mathematica* uses these forms for special purposes. (For details, see the documentation entitled How Modules Work.) Also note that superscripts and subscripts do not become part of a symbol name.

Manipulating Symbols

You can manipulate symbols without associating them with any values.

```
Clear[a, b, c, x]
Solve[a * x^2 + b * x + c == 0, x]
```

$$\left\{\left\{x \to \frac{-b-\sqrt{b^2-4\,a\,c}}{2\,a}\right\},\ \left\{x \to \frac{-b+\sqrt{b^2-4\,a\,c}}{2\,a}\right\}\right\}$$

Note how we first used `Clear` to clear any definitions that might have been assigned to these symbols.

Setting and Unsetting the Values of Variables

When we introduce symbols as variable names, and we often wish to assign values to them. Use a single equal sign (the assignment operator) to define a symbol to be equivalent to the value of a defining expression.

```
Clear[a, b, c]
a = b + c
```

```
b + c
```

The equals sign is actually shorthand for the `Set` command, so the following is equivalent. (But we usually use the equals sign.)

```
Clear[a, b, c]
Set[a, b + c]
```

```
b + c
```

Notice that an assignment expression has a value, produced by the evaluation of the right hand side. Assignment returns this value, and it is displayed as output. You can suppress that output by putting a semi-colon at the end of the line.

```
a = b + c;
```

After such an assignment, the defined symbol will simply be replaced in subsequent expressions by its definition. (We have associated a “rewrite rule” with the symbol `a`, which will be applied whenever *Mathematica* encounters `a`.)

```
a * a
```

```
(b + c)^2
```

An assignment persists until you make a new assignment or `Unset` the symbol. In the latter case, the symbol still exists but no longer refers to its previous definition. Mathematica provides the shorthand `=.` for `Unset`.

```
Clear[a, b, c]
a = b + c;  (* Set the value of `a` *)
a * a
a =.   (* Unset `a`, which now has no value *)
a * a
```

```
(b + c)^2
```

```
a^2
```

Advanced: Once you create a symbol, it persists. When we `Unset` a symbol, we do not remove it from Mathematica’s list of recognized symbols. We can see this list by using the `Names` command. If this is for some reason unacceptable, we can `Remove` the symbol.

```
a00 = 5;
Unset[a00] (* `a00` is still in the list of names: *)
MemberQ[Names["Global`*"], "a00"]
Remove[a00](* `a00` is no longer in the list of names: *)
MemberQ[Names["Global`*"], "a00"]
```

```
True
```

```
False
```

Multiple Assignment and Clear

You can make many assignments at once by putting comma-separated variable names and assigned values in braces. Since the right hand side of an assignment is evaluated before the assignment takes

place, you can use multiple assignment to swap variable values.

```
{a01, a02} = {1, 2};     (* set both values *)
{a01, a02}               (* show both values *)
{a01, a02} = {a02, a01}; (* swap values *)
{a01, a02}
```

```
{1, 2}
```

```
{2, 1}
```

You can `Unset` many symbols by providing them as arguments to `Clear`.

```
Clear[a01, a02]
{a01, a02}
```

```
{a01, a02}
```

Advanced: Use ClearAll instead of Clear if you want to clear Attributes and Options associated with a symbol. You can also Remove a symbol, but this has subtle repercussions. (It will affect all your previous definitions using that symbol.)

Advanced: You can use the OwnValues command to examine the assignment to a variable. We see that assignment has created a rule associated with that variable. For example, if we have made an assignment to `a` but not to `b` we find a rule in `OwnValues[a]` but none in `OwnValues[b]`.

```
Clear[a, b, c]
a = b + c
OwnValues[a]
OwnValues[b]
```

```
b + c
```

```
{HoldPattern[a] :> b + c}
```

```
{}
```

Delayed Assignment

We have seen that assignment (`Set`) immediately evaluates the right hand side and assigns this value to the left hand side. The evaluation of the right-hand side is done once.

```
Clear[a, b];
a = 0; b = a; a = 1; b
```

```
0
```

Sometimes we want different behavior: the right hand side is not evaluated until the left hand side symbol is used. *Mathematica* allows us to achieve this with the `SetDelayed` command, or the equivalent := operator. The evaluation of the right hand side is done anew each time the left-hand side symbol appears.

```
a = 0; b := a; a = 1; b
```

```
1
```

Here is another way to see the different between assignment and delayed assignment.

```
x1 = RandomInteger[100]
{x1, x1, x1, x1} (* the same each time *)
x2 := RandomInteger[100]
{x2, x2, x2, x2} (* can differ each time *)
```

```
44
```

```
{44, 44, 44, 44}
```

```
{9, 40, 90, 80}
```

Advanced: Notice the difference in the `OwnValues` of `b` in the following cases. Related to this, in contrast to `Set`, the `SetDelayed` command returns a `Null` value.

```
ClearAll[a, b1, b2, b3]
{a = 2, b1 = a};
OwnValues[b1]
{a := 2, b2 = a};
OwnValues[b2]
{a := 2, b3 := a};
{OwnValues[b1], OwnValues[b2], OwnValues[b3]}
{b1, b2, b3}
a = 3
{b1, b2, b3}
```

```
{HoldPattern[b1] :↦ 2}
```

```
{HoldPattern[b2] :↦ 2}
```

```
{{HoldPattern[b1] :↦ 2}, {HoldPattern[b2] :↦ 2}, {HoldPattern[b3] :↦ a}}
```

```
{2, 2, 2}
```

```
3
```

```
{2, 2, 3}
```

Equality vs. Assignment

Note that *Mathematica* has several different uses of the equals sign.

```
x1 = 2 (* assignment *)
x2 := Random[] (* delayed assignment *)
1 == 1.0 (* equality testing *)
1 === 1.0 (* identity testing *)
```

```
2
```

```
True
```

```
False
```

Variable Scope

By default *Mathematica* symbols have global scope: they are available anywhere in your notebook. For users accustomed to traditional programming languages, this has some surprising implications, which trace to our ability to manipulate undefined symbols in *Mathematica* expressions. For example, without defining *x* or *y* we can write utility as

```
Clear[x, y]
U = x^0.5 y^0.5
```

$$x^{0.5}\, y^{0.5}$$

Mathematica uses this expression to define *U* but *x* and *y* remain undefined. Moreover, we can then subsequenctly assign values to these varaibles, and it affects the values of *U*.

```
x = 20; y = 30;
U
```

```
24.4949
```

If we change the values of *x* or *y*, it changes the value of *U*.

```
x = 50;
U
```

```
38.7298
```

If we clear the values of *x* and *y*, then we return to a definition of *U* in terms of undefined variables.

```
Clear[x, y]
U
```

$$x^{0.5}\, y^{0.5}$$

The default global scope is often convenient, but it does mean that if you re-evaluate an expression earlier in a notebook after you change (later in the notebook) any variable in the expression, this will affect the evaluation of that expression. That is, the order of occurence in a notebook need not signal the order in which expressions have been evaluated. (You can pick Evaluate Notebook from the Evaluation menu if you want to evaluate all the evaluatable cells in order.) For this and other reasons, you may want to ensure that some variables are local to an expression. You can accomplish this with the Module command. (For more details, see the *Mathematica* documentation entitled How Modules Work.)

```
t = 5;
Module[{t = 0}, t += 1; Print[t]]
Print[t]
```

```
1
```

```
5
```

Comparison

We can test equality between symbolic expressions:

```
Clear[a, b]
a = b;  (* assignment *)
a == b   (* equality test *)
```

```
True
```

Symbolic equations may not automatically be simplified. In this case the original equation is returned.

```
Sin[θ]^2 + Cos[θ]^2 == 1
```

```
Cos[θ]^2 + Sin[θ]^2 == 1
```

We may be able to force simplification with the `Simplify` command (or the more costly `FullSimplify` if necessary).

```
Simplify[%]
```

```
True
```

If we restrict their values, we may even be able to simplify to a boolean value size comparisons between symbols that have not been assigned numerical values. Note how the relational operators get reused to impose relations, not only to test for them.

```
Simplify[x < y, x < 3 && y > 4]
```

```
True
```

Warning: Expressions that appear identical may not be identical, if they have side effects. For example, the `Increment` command increases the value of a symbol by 1 (and returns the old value). (There is a puzzle as to how a command can directly change the value of its argument, which we will take up later when discussing the `HoldAll` attribute.) We can append `++` to a symbol as a shorthand for `Increment`.

```
a = 1;
a++ == a++
a
```

```
False
```

```
3
```

Help

Mathematica's basic help facility is its `Information` command, which provides help for any symbol. The help begins with a basic "usage message" and the provides some details about the symbol, including any options.

```
a = b + c;
Information[a]
```

Info-aeb39697-4ef6-44ec-96c5-bedbc86d17f4

```
Global`a
```

Info-aeb39697-4ef6-44ec-96c5-bedbc86d17f4

```
a = b + c
```

Here we learn that `a` is a global symbol that has been set equal `b+c`.

We often want information about Mathematica commands. To make this much shorter to type this as an "input escape", prepending two question marks to the symbol.

```
?? Information
```

Info-3cd93d23-7d8e-44c4-9bc3-f96744387842

Information[*symbol*] prints information about a symbol. ≫

Info-3cd93d23-7d8e-44c4-9bc3-f96744387842

```
Attributes[Information] = {HoldAll, Protected, ReadProtected}

Options[Information] = {LongForm → True}
```

We often only want the usage message, which is available as an option by specifying LongForm->False. To specify that we do not want the long-form information, we can alternatively use a single question mark instead of two.

```
? Information
```

Info-40bca079-4b37-4ce1-9f77-cc24162448b4

Information[*symbol*] prints information about a symbol. ≫

Additional Observations

Warning: As you start working in your Notebooks, remember that symbol definitions are global. Computations are done by the *Mathematica* kernel. If you open multiple notebooks during a single *Mathematica* session, the notebooks will share a single kernel by default. (You can change this.) This means that all your open notebooks are sharing all your global definitions!

Aside: to completely remove all your global symbols, it is safest just to use the Quit[] command. This quits the kernel and then starts a new *Mathematica* session.

Advanced: The front end and the kernel talk to each other via *MathLink*, a communications protocol. Programmers can use *MathLink* to communicate with *Mathematica* from their programs or other applications.

Contexts

Every *Mathematica* symbol has a context, which is part of its full name. When you create a new symbol

at a notebook prompt, it is ordinarily part of the Global` context. You can determine the context of a symbol with the `Context` command.

```
Context[a]
```

```
Global`
```

IMPORTANT: by default, symbols in the global context are available to all your open notebooks! You can set a Notebook's Default Context via the Evaluation menu.

A context name always ends with a backtick, called a context mark. You can create a new context at any time simply by including a new context name when defining a symbol.

```
Context[my`a]
Context[yr`a]
```

```
my`
```

```
yr`
```

```
a + my`a + yr`a
```

```
my`a + yr`a + b + c
```

For more information, see the *Mathematica* documentation entitled Contexts.

Entering Math

Typing Math in Text Cells

Often we wish to include inline math in a Text cell. We do this by creating an inline-math cell, usually by pressing Ctrl+9. A lightly colored box appears, and you can begin typing your math. When you are done typing your math, press ctrl-0 to exit. On a standard American keyboard, the numbers 9 and 0 are associated with opening and closing parentheses, which provides a nice mnemonic for math entry. Notice that *Mathematica* applies different formatting to the inline-math cell than it does to ordinary text. Here is an example: $y = f(x)$.

In order to type mathematics, you will also want to learn how to enter special symbols and formats. One simple approach is to pick from the menus Palettes > Basic Math Assistant, which gives point and click access to a useful collection of symbols and typesetting formats. For faster entry, you will want to learn keyboard shortcuts. *Mathematica* allows use of the escape (esc) key to delimit keyboard shortcuts (called aliases) to special symbols. You can also use full names for the symbols, surrounded by brackets, and preceded by a backslash. So we can enter an α either as [esc]a[esc] or as \[Alpha], and we can enter $\int$ as either [esc]int[esc] or as \[Integral]. Finally, we often want to type superscripts or subscripts: rather than rely on the Basic Math Assistant, it is good to learn the keyboard shortcuts: Ctrl+6 (or alternatively, Ctrl+^ on an American keyboard) and Ctrl+- (or equivalently, Ctrl+_ on an American keyboard) bring up the superscript and subscript templates. Now you can type expressions such as $y = \alpha_1 x^2$ in your Text cells.

In the Basic Math Assistant, if you hover your mouse over a special format, *Mathematica* will display the keyboard shortcut for that format.

Some online resources:
http://reference.wolfram.com/mathematica/guide/GreekLetters.html
http://reference.wolfram.com/mathematica/guide/SpecialCharacters.html
http://reference.wolfram.com/mathematica/tutorial/KeyboardShortcutListing.html

HoldForm

When we want *Mathematica* to evaluate a mathematical expression, we type that expression into a cell that has the default `Input` style. When that cell is active, we can evaluate the expression by pressing `Shift+Enter`. The result of the evaulation will display in a new cell, which is given an `Output` style. Here is an example from the Wolfram documentation.

```
HoldForm[Integrate[x^2 E^-x^2, x]] == Integrate[x^2 E^-x^2, x]
```

$$\int x^2\, e^{-x^2}\, dx == -\frac{1}{2}\, e^{-x^2}\, x + \frac{1}{4}\sqrt{\pi}\ \text{Erf}[x]$$

Note the use of HoldForm, which prevents the evaluation of the expression on the left. As a result we can display both the expression we want to evaluate and its evaluation. This is the kind of thing we might want to include in our text. *Mathematica* distinguishes display math from inline math. Display math is placed in a separate cell with style DisplayFormula or DisplayFormulaNumbered. This is where you will usually type stand-alone equations and expressions involving two-dimensional structures, such as matrices. So let us copy it into a new cell, which we give the DisplayFormula style.

$$\int x^2\, e^{-x^2}\, dx == -\frac{1}{2}\, e^{-x^2}\, x + \frac{1}{4}\sqrt{\pi}\ \text{Erf}[x]$$

Matrices in a Display Formula

Usually you will want to type a matrix formula directly into a cell having the DisplayFormula style. Here is one way. Begin by creating a DisplayFormula cell and entering your matrix equation.

```
{{a11, a12}, {a21, a22}} {{x1}, {x2}} == {{b1}, {b2}}
```

Next, press Ctrl+Shift+t to change it to traditional display:

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

Display formula are not automatically center-aligned. To change the alignment, pick from the menu Format > Text Alignment > Align Center.

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

Advanced: Entering Math as LaTeX

If you are familiar with LaTeX, you may at time find it more convenient to enter math expressions using LaTeX notation. *Mathematica* allows you to do this with ToExpression[“inputstring”,TeXForm], where any backslashes in the input string must be doubled.

```
ToExpression["\\sqrt{2.0}", TeXForm]
```

```
1.41421
```

Since LaTeX notation is not always unambiguous, *Mathematica* can be very touchy about how you use it. For example, ToExpression[“\\int_0^3 2*x dx”,TeXForm] will fail. But if we force a space, *Mathematica* will recognize (and evaluate) the expression.

```
ToExpression["\\int_0^3 2*x \, dx", TeXForm]
```

```
9
```

If you want to see the LaTeX that *Mathematica* associcates with an expression, use TeXForm.

```
TeXForm[Integrate[f[x], {x, 0, 3}]]
```

```
\int_0^3 f(x) \, dx
```

Drawing with Graphics Objects

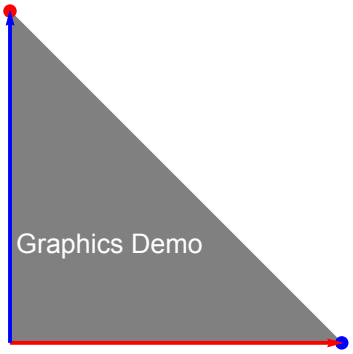
Interactive Drawing

Mathematica allows for interactive drawing in your notebooks. From the *Mathematica* menus pick Insert>Picture>NewGraphic, or just press Ctrl+1, to create a new empty graphic. Then press Ctrl+D to bring up the Drawing Tools palette. At the top of the palette you can find a variety of objects to place in your drawing. You can set the properties (e.g., color and line thickness) before drawing the object, or you can left-click the object after drawing it and change its properties. (For more details, see the *Mathematica* documentation entitled Graphics Interactivity & Drawing and the documentation entitled Drawing Tools.)

However, for ease of modification and ready replication, it is a good habit to draw with code rather than with the interactive tools.

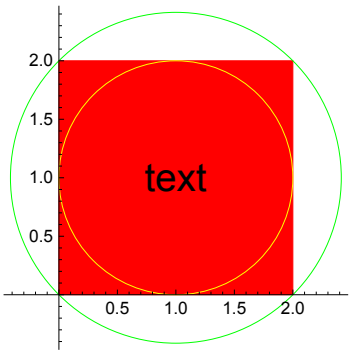
Example: Polygon, Point, Arrow, and Text

```
p1 = {0, 0}; p2 = {1, 0}; p3 = {0, 1};
g1 = Graphics[
   {
    {Gray, Polygon[{p1, p2, p3}]},
    {PointSize[Large], Red, Point[p3]},
    {PointSize[Large], Blue, Point[p2]},
    {Thick, Red, Arrow[{p1, p2}]},
    {Thick, Blue, Arrow[{p1, p3}]},
    {White, Text[Style["Graphics Demo", 14], {0.3, 0.3}]}
   },
   ImageSize → Small]
```



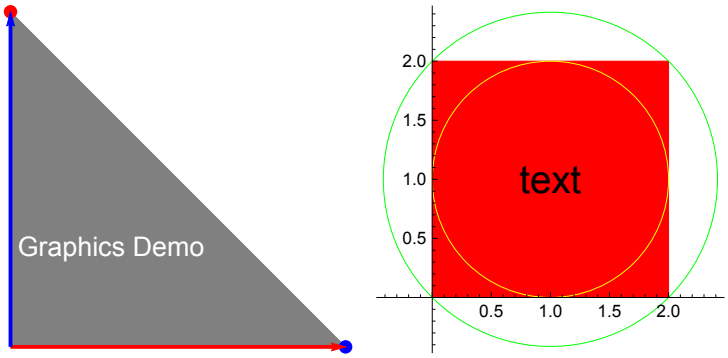
Example: Rectangle, Circle, Axes

```
g2 = Graphics[{
    {Red, Rectangle[{0, 0}, {2, 2}]},
    {Yellow, Circle[{1, 1}, 1]},
    {Green, Circle[{1, 1}, Sqrt[2]]},
    Text[Style["text", 20], {1, 1}]
   }, Axes → True, ImageSize → Small]
```



Example: Combining Graphics Objects

```
g3 = GraphicsRow[{g1, g2}]
Export["c:/temp/temp.pdf", g3]  (* WARNING: this will overwrite temp.pdf *)
```




```
c:/temp/temp.pdf
```

Advanced: Programmatic Drawing

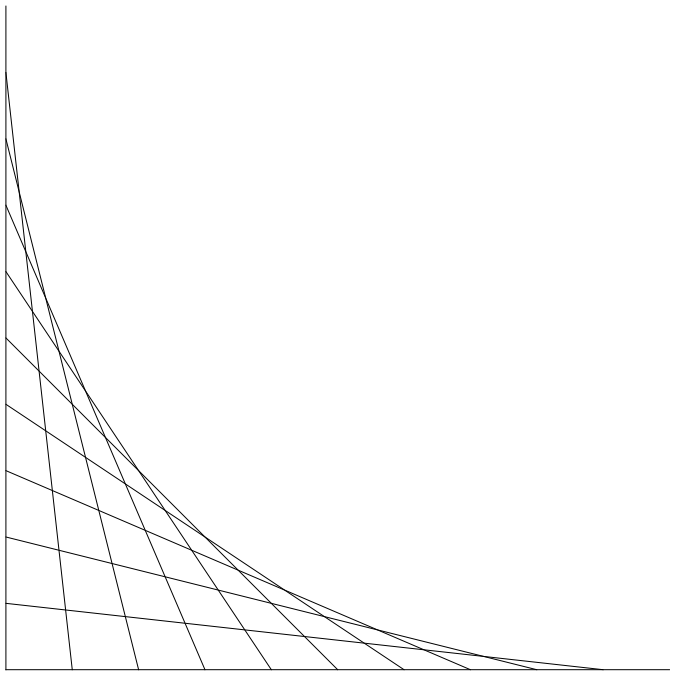
We can also generate graphics programmatically, which is often more useful. (It is more precise and more replicable.) For two-dimensional graphics, we create a Graphics object from a list of graphics primitives such as Point, Line, Arrow, and Text. In this context, point is produced from a list of two coordinates, a line or arrow is produced from a list of two or more points.

As an example, let us use `Array` to create a list of pairs of pairs of numbers. Each pair of numbers represents a point. Each pair of pairs therefore represents a line. The `Line` command can accept this array as a description of a collection of lines, and the result can be drawn by the `Graphics` command.

```
Array[{{0, 10 - #}, {#, 0}} &, 11, 0]
Line[%]
Graphics[%]
```

```
{{{0, 10}, {0, 0}}, {{0, 9}, {1, 0}}, {{0, 8}, {2, 0}},
 {{0, 7}, {3, 0}}, {{0, 6}, {4, 0}}, {{0, 5}, {5, 0}}, {{0, 4}, {6, 0}},
 {{0, 3}, {7, 0}}, {{0, 2}, {8, 0}}, {{0, 1}, {9, 0}}, {{0, 0}, {10, 0}}}
```

```
Line[{{{0, 10}, {0, 0}}, {{0, 9}, {1, 0}}, {{0, 8}, {2, 0}},
  {{0, 7}, {3, 0}}, {{0, 6}, {4, 0}}, {{0, 5}, {5, 0}}, {{0, 4}, {6, 0}},
  {{0, 3}, {7, 0}}, {{0, 2}, {8, 0}}, {{0, 1}, {9, 0}}, {{0, 0}, {10, 0}}}]
```



See http://reference.wolfram.com/mathematica/guide/GraphicsObjects.html for more graphics objects. See http://reference.wolfram.com/mathematica/tutorial/GraphicsDirectivesAndOptions.html for graphics directives and options.

Sharing Notebooks

Notebook files are saved with a .nb extension. These are readily shared with anyone who has *Mathematica*. However *Mathematica* allows you to share both static and dynamic content with others.

The most useful format for sharing static content is probably the portable document format (PDF). First, save your notebook. Then, pick from the *Mathematica* menus File>SaveAs>Save as Type> PDF Document. (Alternatively, you can Export the EvaluationNotebook.)

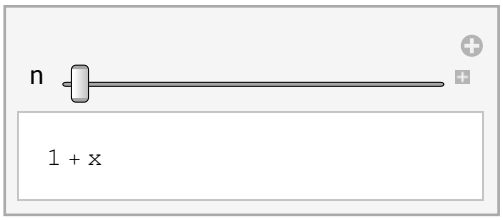
For sharing dynamic content with people who do not have *Mathematica*, in version 8 and above you can save your notebook in the computable document format (CDF). These documents can be opened in the free Wolfram CDF Player. The static content of an ordinary notebook is viewable in the CDF player, but

the CDF format additionally produces Manipulate objects that are fully interactive in the player. (Also supported are Dynamic and DynamicModule.)

Manipulate Example

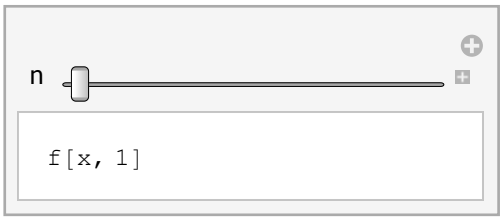
Here is a simple example of the use of Manipulate.

```
Clear[x]
Manipulate[Expand[(1 + x)^n], {n, 1, 5, 1}]
```



If your Manipulate object relies on defined symbols, you will need to use the SaveDefinitions option to ensure interactivity in the CDF player. Here is an example that uses a function definition. (We give a fuller discussion of function defintion in a later section.)

```
ClearAll[f, x, n]
f[x_, n_] := (1 + x)^n
Manipulate[Expand[f[x, n]], {n, 1, 5, 1}, SaveDefinitions → True]
```



Exercise

Create a new notebook and then do the following.
(a) From the menus, pick Format > Style Title and then enter the title “My First Notebook”.
(b) Create a Section cell and enter the subtitle “Text and Expressions”.
(c) Create a Text cell, enter your name, and right justify the cell contents. (Look under the Format>Alignment menu.)
(d) Create a Section cell and enter "Just Text".
(e) Create a Text cell and enter the text “My nicely formatted text cell.”.
(e) Create a Section cell and enter the section heading “Just Computation”.
(f) Create an Input cell (the default style), enter an expression, and evaluate it.

Some Useful Resources

Remember, the *Mathematica* documentation is generally excellent, and it is available both in the Note-

book and online. Nevertheless, here are a few supplementary resources.

- Input Syntax: http://reference.wolfram.com/mathematica/tutorial/InputSyntax.html
- Herbert Halpern's introduction: https://math.uc.edu/~halpern/Mathematicafall09/Ho/Mathematicabook24.pdf
- Wolfram's *Mathematica* tutorials: https://www.wolfram.com/learningcenter/tutorialcollection/
- Pitfalls for *Mathematica* beginners: http://mathematica.stackexchange.com/questions/18393/what-are-the-most-common-pitfalls-awaiting-new-users
- Leonid Shifrin's *Mathematica* Programming: http://www.mathprogramming-intro.org/

Functions: Some Basic Considerations

Mathematically, a function maps elements of one set to elements of another set. This is a very general concept. Computationally, functions are usually much more restricted in scope. For example, a function will often specify a particular transformation of an "input argument" into a new value that the function returns.

Pure Functions

Suppose that we want a function to return x^2 for any input x. Mathematica allows us to create this in a very natural way with the `Function` command: first we give names to the inputs. (The names are called parameters or formal parameters; the input values are called arguments or actual parameters.) Then we use these names to specify the transformation of input(s) into output(s).

```
f1 = Function[x, x^2]
```

```
Function[x, x^2]
```

Note that the formal parameters are names that are local to the function definition. You do not need to worry about any global variable having the "same" name.

Our function can now be applied to different arguments, numerical or symbolic.

```
f1[2]
f1[s]
```

```
4
```

```
s^2
```

The basic considerations are that simple. If we want a function of more than one variable, we can provide a list of parameter names.

```
f2 = Function[{x, y}, x^2 + y^2]
f2[2, 3]
```

```
Function[{x, y}, x^2 + y^2]
```

```
13
```

We bound the names `f1` and `f2` to our functions so that we could use them later. But these functions

are usable even if we do not name them. Here is a simple example:

```
Function[x, x^2][2]
```

```
4
```

Anonymous functions are useful whenever a function will only be used once, so that naming it is a needless expense. (We will see a number of examples of this later on.)

Alternative Notation for Pure Functions

Mathematica provides a convenient shorthand equivalent using a familiar mathematical notation. (You can produce the “mapsto” arrow as `[Esc]fn[Esc]`.)

```
f3 = x ↦ x^2
f3[2]
f3[s]
```

```
Function[x, x^2]
```

```
4
```

```
s^2
```

In our function definitions up to now, we have given names to the function parameters. These names are local to the function definition, so this is usually harmless and serves as an aid to reading. But we are not required to name our arguments. Instead we can refer to them by their slot numbers. This allows us to skip naming the formal parameters and instead refer to them by their default labels: #1, #2, etc. (Also, # is equivalent to #1.)

Here is a function pure function with one formal parameter and another with two formal parameters.

```
Function[#^2]
Function[#1^2 + #2^2]
```

```
#1^2 &
```

```
#1^2 + #2^2 &
```

Notice that *Mathematica* represents these function definition in an alternative notation, using a postfixed ampersand. We are also free to use this notation directly. It is a bit more compressed but perhaps a little less readable. It is a very common Mathematica usage. However, especially if you will share your code, it is worth asking whether the gain from the compressed syntax offsets the cost of reduced readability.

Local Variables

Often we want to declare local variables inside our function definition. In *Mathematica* we do this with the `Module` command, which takes a list of variable names as its first argument, which are then treated as local in the expression that is its second argument. The second argument can be a compound expression, where subexpressions are separated by semicolons. The last expression evaluated is the value of the module.

```
Function[x, Module[{s, c}, s = Sin[x]; c = Cos[x]; s^2 + c^2]]
%[Pi / 2]
```

```
Function[x, Module[{s, c}, s = Sin[x]; c = Cos[x]; s^2 + c^2]]
```

```
1
```

`Module` also allows the local variables to be initialized when declared.

```
Function[x, Module[{s = Sin[x], c = Cos[x]}, s^2 + c^2]]
%[Pi / 4]
```

```
Function[x, Module[{s = Sin[x], c = Cos[x]}, s^2 + c^2]]
```

```
1
```

Advanced: Argument Passing

The default behavior of Mathematica is that function arguments are passed as the value of the expression. If you forget about this, you are likely to run into the following error message:

Set::setps {...} in the part assignment is not a symbol

Here is a simple way to produce this error:

```
{0}[[1]] = 1;
```

```
Set::setps : {0} in the part assignment is not a symbol. >>
```

However the following does not produced this error:

```
x = {0}; x[[1]] = 1; x
```

```
{1}
```

The difference is that `Set` changes the value associated with a symbol (in this case, `x`). You need a symbol to associate with the value.

Unfortunately, the place where you are most likely to encounter this error is slightly more obscure: if you pass a symbol to a function, it will be evaluated before the function body is exected. (There are ways to work around this, e.g. using `HoldFirst`, which we do not discuss.)

```
Clear[x, s]
s = {0}
Function[x, x[[1]] = 1]
%[s]  (* problem: s will be evaluated before function is executed *)
```

```
{0}
```

```
Function[x, x[[1]] = 1]
```

```
Set::setps : {0} in the part assignment is not a symbol. >>
```

```
1
```

Advanced: Closures

Functions can return functions, and the returned functions can close over the local variables of the creating function. (A closure can “recall” the environment it was created in.)

```
ClearAll[plusn]
plusn = n ↦ (x ↦ x + n)
plus2 = plusn[2]
plus2[5]
```

```
Function[n, Function[x, x + n]]
```

```
Function[x$, x$ + 2]
```

```
7
```

We see that `plus2` “recalls” that, when it was created, *n* had the value 2.

Note *Mathematica's* special use of the dollar sign for internally defined variable names. Do not use the dollar sign in your own variable names.

Functions via Pattern Matching and Delayed Evaluation

A popular approach to function definition is through a delayed-evaluation assignment. The function name and underscore-tagged formal parameters appear to the left, then the delayed evaluation assignment operator `:=`, and finally the function definition on the right. Note that because we use delayed evaluation, this definition simply defines a symbol and does not generate any output.

```
ClearAll[xsq03]
xsq03[x_] := x^2
xsq03[33]
```

```
1089
```

We can see this difference when we look at the Head or FullForm of the different expressions.

```
Map[Head, {xsq01, xsq02, xsq03}]
Map[FullForm, {xsq01, xsq02, xsq03}]
```

```
{Symbol, Symbol, Symbol}
```

```
{xsq01, xsq02, xsq03}
```

Nevertheless, as we have seen, all three approaches give us a usable “function”.

There are subtle advantages to the different forms. A pure function can be readily added to an expression that needs a function “on the fly”. However, the delayed evaluation syntax allows easy inclusion of default values for a function paramater, given after a colon. (Pattern matching can also be used for type checking, but we do not cover that here.)

```
xsq04[x_: 2] := x * x
xsq04[]
```

```
4
```

Advanced: More Closures

We can also return closures from functions defined by delayed evaluation.

```
ClearAll[plusn]
plusn[n_] := (x ↦ x + n)
plus2 = plusn[2] (* produces a closure *)
plus2[5]
```

```
Function[x$, x$ + 2]
```

```
7
```

We see that `plus2` “recalls” that, when it was created, *n* had the value 2.

Note again *Mathematica's* special use of the dollar sign for internally defined variable names.

Lists

The list is a fundamental *Mathematica* data type. We use it to represent sets, vectors, and matrices.

Quick Introduction

A list is just an ordered collection of items.

Enumeration

We can create a list by enumerating comma-separated elements within curly braces.

```
lst01 = {1, 2, 3}
lst02 = {4, 5, 6}
```

```
{1, 2, 3}
```

```
{4, 5, 6}
```

The numbers are the elements of the list. We can join lists to make a bigger list:

```
Join[lst01, lst02]
```

```
{1, 2, 3, 4, 5, 6}
```

A list can be empty:

```
lst = {}
```

```
{}
```

A list can contain anything as its elements, including other lists. For example, we represent a matrix as a list of lists.

```
mA = {lst01, lst02}
```

```
{{1, 2, 3}, {4, 5, 6}}
```

We can use the `MatrixForm` command to get a more intuitive visual display of a matrix.

```
MatrixForm[mA]
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

Note: `{a,b,c}` is just shorthand for `List[a,b,c]`. We can use `FullForm` to see this:

```
Clear[a, b, c]
FullForm[{a, b, c}]
```

```
List[a, b, c]
```

List Arithmetic

Arithmetic operations on lists are elementwise. So is exponentiation.

```
lst01 + lst02
lst01 - lst02
lst01 * lst02
lst01 / lst02
lst01 ^ lst02
```

```
{5, 7, 9}
```

```
{-3, -3, -3}
```

```
{4, 10, 18}
```

$$\left\{\frac{1}{4}, \frac{2}{5}, \frac{1}{2}\right\}$$

```
{1, 32, 729}
```

Scalar operations are also possible.

```
lst01 + 2
lst01 - 2
lst01 * 2
lst01 / 2
lst01 ^ 2
```

```
{3, 4, 5}
```

```
{-1, 0, 1}
```

```
{2, 4, 6}
```

$$\left\{\frac{1}{2}, 1, \frac{3}{2}\right\}$$

```
{1, 4, 9}
```

```
2 + lst02
2 - lst02
2 * lst02
2 / lst02
2 ^ lst02
```

```
{6, 7, 8}
```

```
{-2, -3, -4}
```

```
{8, 10, 12}
```

$$\left\{\frac{1}{2}, \frac{2}{5}, \frac{1}{3}\right\}$$

```
{16, 32, 64}
```

Indexing

Using double brackets, we can access the elements of a list with a unit-based index. To index backwards from the end of the list, use negative indexes.

```
lst01 = {1, 2, 3}
lst02 = {4, 5, 6}
lst01[[1]]
lst01[[-1]]
```

```
{1, 2, 3}
```

```
{4, 5, 6}
```

```
1
```

```
3
```

Lists can contain lists. We can index the outer list to get the inner lists. We can then index the inner lists to get their elements. There is also a shorthand for this nested indexing: separate the levels at which you are indexing by commas.

```
lstlst = {lst01, lst02}
lstlst[[1]][[2]]
lstlst[[1, 2]]
```

```
{{1, 2, 3}, {4, 5, 6}}
```

```
2
```

```
2
```

Note: `lst[[1]]` is just a shorthand for `Part[lst,1]`. For more details about indexing, see the documentation for `Part`.

Digression on Summing and Accumulating (Fold and FoldList)

Use `Total` to sum up the elements of a list.

```
Total[{a, b, c}]
```

```
a + b + c
```

This is equivalent to using the list elements as arguments to `Plus`.

```
FullForm[Total[{a, b, c}]]
```

```
Plus[a, b, c]
```

We are going to digress a bit and explore this observation in some detail. A *Mathematica* list has a "head" of list, which is available as part 0 of the list.

```
Head[{a, b, c}]
{a, b, c}[[0]]
```

```
List
```

```
List
```

Mathematica allows us to replace the head, using the `Apply` command (equivalent to the `@@` operator). The Apply command simply replaces the "head" of an expression with a specified function.

```
Clear[f, a, b]
Apply[f, {a, b}]
```

```
f[a, b]
```

We can use `Apply` to replace the "head" of a list with `Plus` in order to find the total sum of list elements. Recall that *Mathematica* provides the `@@` shorthand for `Apply`, which makes this a little easier to write (but perhaps a little less understandable).

```
Apply[Plus, {a, b, c}]
% === Plus @@ {a, b, c}
```

```
a + b + c
```

```
True
```

Now for one more digression. Let us do exactly the same thing using folds. Basically, a fold repeatedly apply a binary operation, using an accumulated value and successive elements of a list. We have to provide an initial value for the accumulator. Folds are common in functional programming languages; showing its functional emphasis, *Mathematica* supports folds.

```
ClearAll[f, x0]
Fold[f, x0, {x1, x2, x3}]
```

```
f[f[f[x0, 2], 0.628989], x3]
```

For example, we can form the total sum of list elements by folding `Plus` over the list with a `0` initial value for the accumlator.

```
Fold[Plus, 0, {a, b, c}]
```

```
a + b + c
```

This is a reminder that in *Mathematica* there are often many way to accomplish a single goal. Usually the best choice to make will be the one that you will find easiest to read when you return to your code.

Exercise: use `Fold`, `Times`, and `Range` to produce 10!.

```
Fold[Times, 1, Range[10]] == 10!
```

```
True
```

We can use `Accumulate` to produce the cumulative sum. The last term is the same a folding `Plus` over the list, but we also get intermediate terms.

```
Accumulate[{a, b, c}]
```

```
{a, a + b, a + b + c}
```

`Accumulate` is a special example of folding a list and keeping the intermediate values. This is exactly what the `FoldList` command does. There is one important difference: `FoldList` includes an initial value (that we provide). But we can use `Rest` to discard this initial value.

```
Rest[FoldList[Plus, 0, {a, b, c}]]
```

```
{a, a + b, a + b + c}
```

We can similarly accumulate products.

```
Rest[FoldList[Times, 1, {a, b, c}]]
```

```
{a, a b, a b c}
```

List Creation

Constant Arrays and Ranges

Mathematica provides powerful facilities for the creation and manipulation of lists.

If you plan to repeatedly enumerate a single value, you may find it faster to use `ConstantArray`. We can use `==` to do an equality comparison of lists.

```
list01 = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
list02 = ConstantArray[0, 10];
list01 == list02
```

```
True
```

Integer intervals are a common need, and we usually create them with `Range`. For example, let us create the first twenty natural numbers.

```
nat20 = Range[20]
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20}
```

You can also specify a minimum as well as a maximum value for a range of integers. Note that the

range is inclusive: it contains the minimum and maximum values. A stepsize can be given as a third argument.

```
Range[5, 20]
Range[5, 20, 2]
```

```
{5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20}
```

```
{5, 7, 9, 11, 13, 15, 17, 19}
```

The arguments to `Range` do not have to be integers.

```
Range[1.5, 10.5, 0.5]
```

```
{1.5, 2., 2.5, 3., 3.5, 4., 4.5, 5., 5.5, 6., 6.5, 7., 7.5, 8., 8.5, 9., 9.5, 10., 10.5}
```

Prepending and Appending Elements

You can Append or Prepend to a list. These commands return a new list, without changing the value of the old list.

```
row = ConstantArray[0, 8]
Append[row, -1]
Prepend[row, 1]
row
```

```
{0, 0, 0, 0, 0, 0, 0, 0}
```

```
{0, 0, 0, 0, 0, 0, 0, 0, -1}
```

```
{1, 0, 0, 0, 0, 0, 0, 0, 0}
```

```
{0, 0, 0, 0, 0, 0, 0, 0}
```

If you want not to produce a new list but to change an existing list, use `AppendTo` and `PrependTo` instead.

```
row = ConstantArray[0, 8];
AppendTo[row, -1];
PrependTo[row, 1];
row
```

```
{1, 0, 0, 0, 0, 0, 0, 0, 0, -1}
```

Suppose I append a list to an empty list:

```
lst = {}
AppendTo[lst, row]
```

```
{}
```

```
{{1, 0, 0, 0, 0, 0, 0, 0, 0, -1}}
```

Note that lst has one element, which is a list.

```
Length[lst]
```

```
1
```

List Creation using Table

The `Table` command is a more general facility for list creation. It therefore can readily accomplish the same task as `Range`, but slightly more verbosely.

```
Range[20] == Table[i, {i, 20}]
```

```
True
```

The Table command takes as its first argument an expression in a variable and, as its second argument, an “iterator” in that variable. An iterator is just a list with a special format. The iterator may take a few basic forms. The form with only a positive integer stop value, as in `{i,stop}`, will produce numbers natural numbers up to stop. You can also specify a start and stop value, as in `{i,start,stop}`. The default is a unit increment, but you can change that by specifying a third argument, as in `{i,start,stop,increment}`. An alternative is to offer an explicit list of values, as in `{i,mylist}`.

```
Table[0, {5}]
Table[i, {i, 5}]
Table[i, {i, 6, 10}]
Table[i, {i, 1.5, 10.5, 0.5}]
lst = Range[5, 1, -1]; Table[i * i, {i, lst}]
```

```
{0, 0, 0, 0, 0}
```

```
{1, 2, 3, 4, 5}
```

```
{6, 7, 8, 9, 10}
```

```
{1.5, 2., 2.5, 3., 3.5, 4., 4.5, 5., 5.5, 6., 6.5, 7., 7.5, 8., 8.5, 9., 9.5, 10., 10.5}
```

```
{25, 16, 9, 4, 1}
```

The first argument to `Table` can be any expression you want. For example, to produce a list of lists, the first argument can be a list of outputs.

```
tbl = Table[{x, x^2}, {x, 0, 5}]
```

```
{{0, 0}, {1, 1}, {2, 4}, {3, 9}, {4, 16}, {5, 25}}
```

The `Table` command produces nested lists when given multiple iterators. Here for example is a simple list of lists.

```
Table[i * j, {i, {-1, 1}}, {j, 3}]
```

```
{{-1, -2, -3}, {1, 2, 3}}
```

Advanced: Note that `Table` localizes the iterator with dynamic scoping, so expressions matching the iterator variable will be iterated. (I do not recommend relying on dynamic scoping; it is too easy to lose track of it.) Example:

```
y := Sin[i * Pi / 2]
Table[y, {i, 4}]
```

```
{1, 0, -1, 0}
```

List Creation from External Data

Naturally *Mathematica* can build lists from external data. For convenience, *Mathematica* ships with some sample data. Here we simply replicate the example at https://reference.wolfram.com/language/tutorial/ReadingTextualData.html. First we take a look at the data:

```
FilePrint["ExampleData/numbers"]
```

```
11.1   22.2    33.3
44.4   55.5    66.6
```

Then we read it into a list:

```
ReadList["ExampleData/numbers", Number]
```

```
{11.1, 22.2, 33.3, 44.4, 55.5, 66.6}
```

Mathematica also includes web access to a variety of data sets. For example, `CountryData[]` produces a list of countries in the database. We can use this same function to produce a variety of sublists. For example, we query for the membership of the EU, the G8, or the G20. These are returned as lists.

```
listG8 = CountryData["G8"]
```

{ Canada , France , Germany , Italy , Japan , United Kingdom , United States }

Note: The special formatting reflects the fact that the list elements are country entities. Such entities may have many predefined properties. See the documentation for details, but the following example provides a hint at what is possible.

```
us = Entity["Country", "UnitedStates"];
gdp2014 = EntityProperty["Country", "GDP", {"Date" → DateObject[{2014}]}];
pop2014 = EntityProperty["Country", "Population", {"Date" → DateObject[{2014}]}];
us[gdp2014] / us[pop2014]
```

```
$54 025.3 per person per year
```

As a final example, *Mathematica* can of course read standard data formats, such as comma-separated values (CSV) files. To illustrate this, let us create one and then read it in.

```
data = RandomInteger[100, {4, 3}] (* make some data *)
(* choose a filename (it must be safe to overwrite!): *)
fname = FileNameJoin[{$TemporaryDirectory, "temp.csv"}];
Export[fname, data]; (* export the data to file *)
FilePrint[fname] (* look at what we stored *)
newdata = Import[fname]; (* import the stored data *)
newdata == data (* check that the data are unchanged *)
```

```
{{12, 62, 47}, {95, 19, 59}, {19, 4, 78}, {65, 64, 80}}
```

```
12,62,47
95,19,59
19,4,78
65,64,80
```

```
True
```

List Manipulation

Mathematica provides powerful list manipulation facilities. Here we touch on a few of the most common ones.

Accessing and Changing List Elements

You can access list elements with doubled brackets. *Mathematica* uses unit based indexing of the list elements; negative indexes count from the end of the list. Here we make a new list out of elements of the old list.

```
lst01 = Range[4];
{lst01[[1]], lst01[[-1]]}
```

```
{1, 4}
```

Mathematica lists are mutable. We can use indexing on the left side of an assignment to change the value of an element. *Mathematica* effectively copies lists on assignment, so in the following example `lst01` is changed but `lst02` is not changed. (I.e., `lst01` and `lst02` refer to two different lists.) Note the use of a list of indexes to change multiple elements.

```
lst01 = Range[4]; lst02 = lst01;
lst01[[{1, -1}]] = {91, 94};
lst01
lst02
```

```
{91, 2, 3, 94}
```

```
{1, 2, 3, 4}
```

We can retrieve and replace contiguous elements with an index range specified as `start;;stop` or `start;;stop;;step`.

```
x = Range[15]
x[[4 ;; 10]]
x[[4 ;; 10 ;; 2]]
x[[4 ;; 10 ;; 2]] = 0
x
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}
```

```
{4, 5, 6, 7, 8, 9, 10}
```

```
{4, 6, 8, 10}
```

```
0
```

```
{1, 2, 3, 0, 5, 0, 7, 0, 9, 0, 11, 12, 13, 14, 15}
```

We can retrieve and replace multiple parts by using a list of indexes.

```
x = Range[10]
x[[{1, 1, 3, 7}]]
x[[{1, 2, 3}]] = {21, 22, 23}
x
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

```
{1, 1, 3, 7}
```

```
{21, 22, 23}
```

```
{21, 22, 23, 4, 5, 6, 7, 8, 9, 10}
```

Other commands for accessing specified elements include `First`, `Rest`, `Most`, `Last`, `Extract`, `Take`, and `Drop`.

Advanced: Accessing and Changing List Elements

The indexing brackets are actually a shorthand for the `Part` command. But most commonly, we use the the indexing brackets.

```
Clear[lst]
lst[[1]] // FullForm // Quiet
```

```
Part[lst, 1]
```

If we want a closer look at what is involved with this assignment with an index, we can again look at the `FullForm`.

```
Clear[x]
HoldForm[FullForm[x[[1]] = 99]]
```

```
Set[Part[x, 1], 99]
```

Assignment with an index mutates an existing list. When a new list is preferred, use `ReplacePart`, which applies a rule (rather than using `Set`).

```
x = {0};
xnew = ReplacePart[x, 1 → 99]
x (* unchanged! *)
```

```
{99}
```

```
{0}
```

Always creating a new list is safer, because it removes ambiguity about the contents of a list. However, it is computationally more costly, especially if many changes will be sequentially made to a large list. It is up to the usef to make the right trade-offs.

Advanced: Although Mathematica lists are mutable, they must be changed indirectly, using the symbol that refers to the list. (In this case, the symbol `lst01`.) For example, the following produces an error:

```
{0}[[1]] = 1;
```

```
Set::setps : {0} in the part assignment is not a symbol. >>
```

This matters when passing lists to functions, because the default behavior is to evaluate any argument before evaluating the function.

Reversing, Sorting, Accumulating

Lists can be reversed, sorted, and accumulated (as a cumulative sum).

```
myList = RandomInteger[20, 5]  (* a list of 5 random numbers *)
mySortedList = Sort[myList]
myReversedList = Reverse[mySortedList]
myCumulativeSum = Accumulate[mySortedList]
```

```
{3, 7, 12, 0, 2}
```

```
{0, 2, 3, 7, 12}
```

```
{12, 7, 3, 2, 0}
```

```
{0, 2, 5, 12, 24}
```

By using `SortBy`, we can sort a list on arbitrary criteria. For example, we can use `SortBy` to sort the G8 countries by population. (We will reverse the sign on population in order to sort from greatest to least.)

```
popSortedG8 = SortBy[listG8, country ↦ -country[pop2014]]
```

{ United States , Japan , Germany , France , United Kingdom , Italy , Canada }

Map and MapThread

Often we wish to apply some function to every element of a list in order to produce a new list of values. For this we use `Map`, for which `/@` is an alternate infix syntax.

```
ClearAll[f]
Map[f, {x1, x2, x3}]
f /@ {x1, x2, x3}
```

```
{f[2], f[0.769947], f[x3]}
```

```
{f[2], f[0.844339], f[x3]}
```

Recall that *Mathematica* allows the creation of anonymous (unnamed) functions. This practice is very common when using `Map`.

```
Map[x ↦ x^2, Range[20]]
(x ↦ x^2) /@ Range[20]
```

```
{1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400}
```

```
{1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400}
```

Recall that *Mathematica* allows us to designate parameter slots instead of naming our paramters. Here is our list of squares once again, produced with this syntax.

```
Map[#^2 &, nat20]
#^2 & /@ nat20
```

```
{1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400}
```

```
{1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400}
```

However, can does not imply should, and the notation in the last example undercuts readability for most people.

If the function used by `Map` produces a list, then we get a list of lists as our result.

```
Map[ctry ↦ {ctry[[2]], ctry[gdp2014] / 10^12}, popSortedG8]
```

```
{{UnitedStates, $17.419 per year},
 {Japan, $4.60146 per year}, {Germany, $3.85256 per year},
 {France, $2.82919 per year}, {UnitedKingdom, $2.94189 per year},
 {Italy, $2.14434 per year}, {Canada, $1.78666 per year}}
```

`MapThread` generalizes `Map` to functions that have multiple arguments. The arguments are given as a list of lists of values, where the inner lists have a common length.

```
MapThread[f, {{x1, x2, x3}, {y1, y2, y3}}]
```

```
{f[2, y1], f[0.96874, y2], f[x3, y3]}
```

For example, we can make a list of rules like this:

```
MapThread[Rule, {{x1, x2, x3}, {y1, y2, y3}}]
```

```
{2 → y1, 0.989897 → y2, x3 → y3}
```

`MapThread` can do much more than this; see the documentation. To give just one example, we can apply a list of functions to a list of arguments.

```
ClearAll[f, g, h, x, y, z]
MapThread[{f, x} ↦ f[x], {{f, g, h}, {x, y, z}}]
```

```
{f[x], g[y], h[z]}
```

Aside: If a function has already been applied to list of arguments, you can use `Thread` instead of `MapThread`. This can be used to thread equations. Somewhat surprisingly, `Thread` allows an argument to be a constant.

```
Thread[f[{x1, x2, x3}, {y1, y2, y3}]]
Thread[{x1, x2, x3} == {y1, y2, y3}]
Thread[{x1, x2, x3} == 0]
Thread[0 == {y1, y2, y3}]
```

```
{f[2, y1], f[0.629764, y2], f[x3, y3]}
```

```
{2 == y1, 0.491755 == y2, x3 == y3}
```

```
{False, False, x3 == 0}
```

```
{0 == y1, 0 == y2, 0 == y3}
```

Exercise: noting that {x,y} is the same as List[x,y], use `MapThread` or `Thread` to transpose a rectangular list of lists.

```
Thread[{{x1, x2, x3}, {y1, y2, y3}}]
MapThread[List, {{x1, x2, x3}, {y1, y2, y3}}]
```

```
{{2, y1}, {0.13913, y2}, {x3, y3}}
```

```
{{2, y1}, {0.101357, y2}, {x3, y3}}
```

```
gdpG8 = Map[c ↦ c[gdp2014], listG8];
popG8 = Map[c ↦ c[pop2014], listG8];
gdppcG8 = MapThread[{c, g, p} ↦ {c[[2]], Round[g / p]}, {listG8, gdpG8, popG8}]
```

```
{{Canada, $50 600 per person per year}, {France, $44 136 per person per year},
 {Germany, $47 198 per person per year}, {Italy, $35 052 per person per year},
 {Japan, $36 454 per person per year}, {UnitedKingdom, $46 288 per person per year},
 {UnitedStates, $54 025 per person per year}}
```

From Lists of Lists to Formatted Tables

We often want to present data in tabular form. A list of lists can be nicely formatted with `TableForm` or `Grid`.

TableForm

The `Table` command can produce a list of lists, using two expressions. Two-dimensional tables can be nicely formatted with the `TableForm` command. Here is an example.

```
tbl = Table[{x, x^2}, {x, 0, 5}]
TableForm[tbl, TableHeadings → {None, {"x", "x^2"}}]
TableForm[tbl, TableHeadings → {None, {"x", "x^2"}}, TableDirections → Row]
```

```
{{0, 0}, {1, 1}, {2, 4}, {3, 9}, {4, 16}, {5, 25}}
```

x	x^2
0	0
1	1
2	4
3	9
4	16
5	25

x	0	1	2	3	4	5
x^2	0	1	4	9	16	25

Recall that the `Table` command also accepts multiple iterators, producing nested lists of lists. Here is an example.

```
ivals = Range[3];
xvals = Range[0, 10];
tbl = Table[x^i, {i, ivals}, {x, xvals}];
formattedTable =
 TableForm[tbl, TableHeadings → {ivals, xvals}, TableAlignments → Right]
```

	0	1	2	3	4	5	6	7	8	9	10
1	0	1	2	3	4	5	6	7	8	9	10
2	0	1	4	9	16	25	36	49	64	81	100
3	0	1	8	27	64	125	216	343	512	729	1000

We may want our tables to be framed and labeled.

```
ySortedG8 = SortBy[gdppcG8, x ↦ -x[[2]]];  (* sort G8 by GDP per capita *)
table = TableForm[ySortedG8, TableHeadings → {None, {"Country", "GDP p.c."}}];
Labeled[Framed[table], "G8 GDP per capita for 2014"]
```

Country	GDP p.c.
UnitedStates	$54 025 per person per year
Canada	$50 600 per person per year
Germany	$47 198 per person per year
UnitedKingdom	$46 288 per person per year
France	$44 136 per person per year
Japan	$36 454 per person per year
Italy	$35 052 per person per year

G8 GDP per capita for 2014

For additional options, read the documentation for the `Table` command.

Using `Grid` for Two-Dimensional Tables

We can achieve fine formatting control with the `Grid` command. (Read the documentation.) But it still provides simple way to produce a two dimensional display of nested lists.

```
mydata = Partition[Range[15], 3];
Grid[mydata]
```

```
 1   2   3
 4   5   6
 7   8   9
10  11  12
13  14  15
```

We can format gridded data, aligning it and adding dividers.

```
mydata02 = Prepend[mydata, {"header1", "header2", "header3"}];
Grid[mydata02, Alignment → Right, Dividers → {None, 2 → True}]
```

header1	header2	header3
1	2	3
4	5	6
7	8	9
10	11	12
13	14	15

We can even shade specific rows.

```
Grid[Prepend[gdppcG8, {"Country", "GDP p.c."}],
 Alignment → Left,
 Dividers → {None, 2 → True},
 Background → {None, 1 → Lighter[Gray, 0.9]}
]
```

Country	GDP p.c.
Canada	$50 600 per person per year
France	$44 136 per person per year
Germany	$47 198 per person per year
Italy	$35 052 per person per year
Japan	$36 454 per person per year
UnitedKingdom	$46 288 per person per year
UnitedStates	$54 025 per person per year

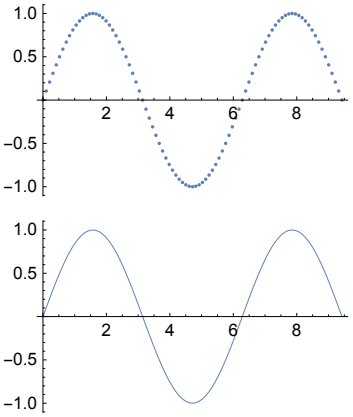
Plots

Plots are an important way to explore data and functions. *Mathematica* provides extensive and powerful plotting facilities. This section provides only a basic introduction to some core plotting functions.

From Lists to Plots

For simple list plotting, use the `ListPlot` and `ListLinePlot` commands. These commands return graph objects that can be manipulated like other objects in *Mathematica.* For example, you can make a list of them.

```
(* make up some data *)
domain = Range[0, 90] * Pi / 30;
sinPoints = Map[x ↦ {x, Sin[x]}, domain];
(* plot the data *)
sinPlotPoints = ListPlot[sinPoints, ImageSize → Small]
sinPlotLine = ListLinePlot[sinPoints, PlotStyle → Thin, ImageSize → Small]
```



Here is an application to the example life-table data that ships with *Mathematica.* We get the data like this:

```
ExampleData[] (* see what categories are available *)
data = ExampleData[{"Statistics", "USLifeTable2003"}];
```

```
{AerialImage, ColorTexture, Dataset, Geometry3D,
 LinearProgramming, MachineLearning, Matrix, NetworkGraph, Sound,
 Statistics, TestAnimation, TestImage, TestImage3D, Text, Texture}
```

We need to know which columns to use. We could try looking at the original data:

```
ExampleData[{"Statistics", "USLifeTable2003"}, "Source"]
```

```
National Vital Statistics Reports, Vol. 54, No 14, April
  19, 2006. http://www.cdc.gov/nchs/data/nvsr/nvsr54/nvsr54_14.pdf
```

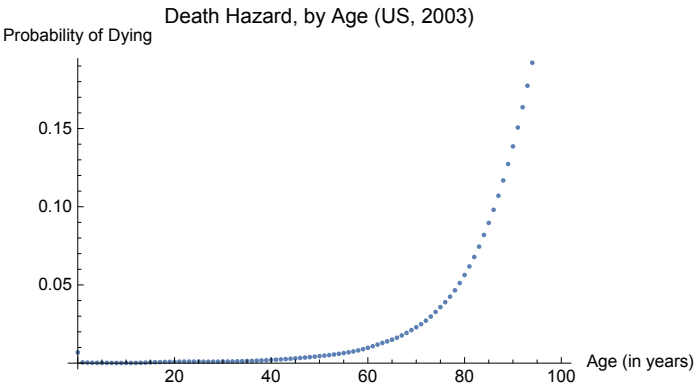
After a little browsing we can figure out what our data must be (Table 1, slightly reformated). However, there is an easier way: *Mathematica*'s example data are stored with helpful information, including column descriptions.

```
ExampleData[{"Statistics", "USLifeTable2003"}, "ColumnDescriptions"]
```

```
{Left endpoint of age interval., Right endpoint of age interval.,
 Probability of dying between AgeLower and AgeUpper.,
 Number surviving to AgeLower., Number dying between AgeLower and AgeUpper.,
 Person-years lived between AgeLower and AgeUpper.,
 Total number of person-years lived above AgeLower.,
 Expectation of life at AgeLower.}
```

We are only going to use the first and third columns to plot the probability of dying against age.

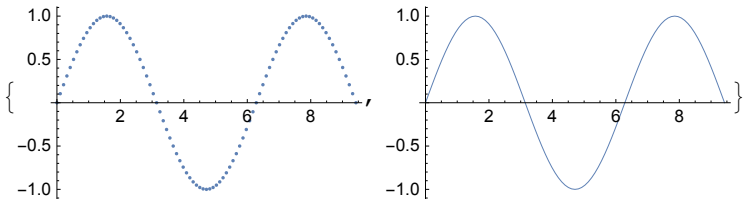
```
plotdata = Map[x ↦ {x[[1]], x[[3]]}, data]; (* get age and death prob *)
ListPlot[plotdata,
 AxesLabel → {"Age (in years)", "Probability of Dying"},
 PlotLabel → "Death Hazard, by Age (US, 2003)"]
```



Charts with Multiple Plots

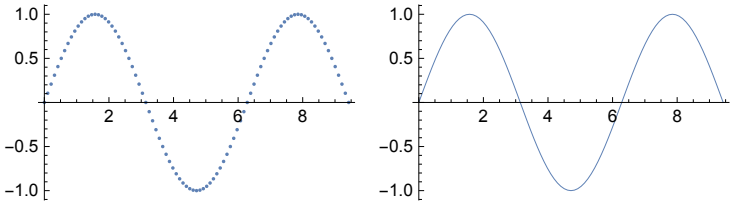
The `ListPlot` and `ListLinePlot` commands return graph objects that can be manipulated like other objects in *Mathematica.* For example, you can make a list of them.

```
sinPlots = {sinPlotPoints, sinPlotLine}
```



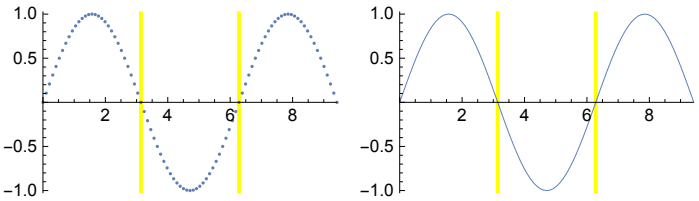
This can be very useful, but for pure display purposes, there are more aesthetic options. Use `GraphicsRow`, `GraphicsColumn`, and `GraphicsGrid` to group list of plots into a single chart with subfigures. For example,

```
GraphicsRow[sinPlots]
```



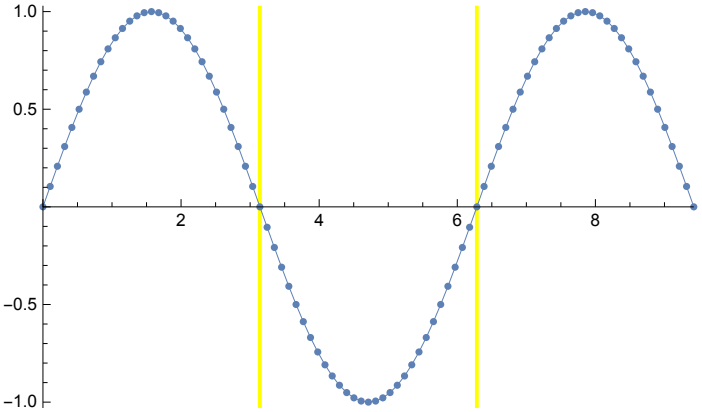
Fine control can be achieved with the many options, which you can see with `Options[ListPlot]`, and *Mathematica* provides help for each option. If you have a style you would like to apply to multiple plots, you can collect the options in a list, say `myoptions`, and then using `Evaluate[myoptions]` when you need it.

```
myoptions = {PlotRange → {{0, 3 * Pi}, {-1.03, 1.03}},
    PlotStyle → Thin,
    GridLines → {{Pi, 2 Pi}},
    GridLinesStyle → Directive[Thick, Yellow]};
sinPlotPoints02 = ListPlot[sinPoints, Evaluate[myoptions]];
sinPlotLine02 = ListLinePlot[sinPoints, Evaluate[myoptions]];
GraphicsRow[{sinPlotPoints02, sinPlotLine02}]
```



Use `Show` to combine multiple plots into a single plot.

```
Show[{sinPlotLine02, sinPlotPoints02}]
```



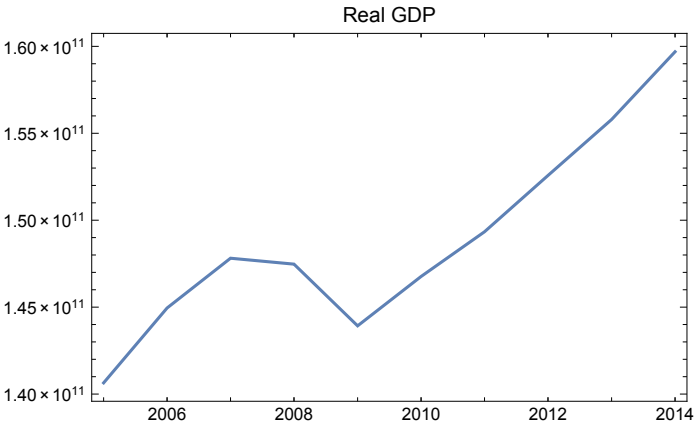
If you need very precise placement of the plot markers, see the discussion at http://mathematica.stackexchange.com/questions/2214/point-renderings-slightly-off-in-mathematica

Plotting Time Series

We can use `DateListPlot` to plot time series. We begin by creating a series of date objects. We then use `Map` over these dates to produces lists of GDP and price-index entity properties, which we use fetch the associated GDP and population time series for one country.

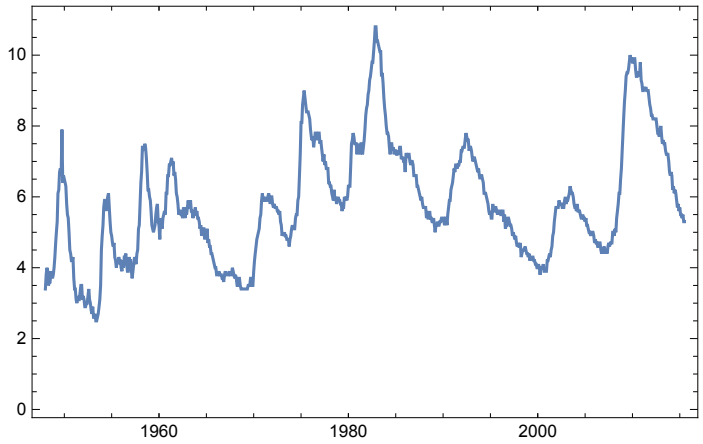
```
dates = Map[DateObject[{#}] &, 2004 + Range[10]];
gdpProps = Map[EntityProperty["Country", "GDP", {"Date" → #}] &, dates];
priceProps = Map[EntityProperty["Country", "PriceIndex", {"Date" → #}] &, dates];


ctry = Entity["Country", "UnitedStates"];
gdps = Map[ctry[#] &, gdpProps];
prices = Map[ctry[#] &, priceProps];
DateListPlot[Transpose[{dates, gdps / prices}], PlotLabel → "Real GDP"]
```



Exercise: Read the *Mathematica* documentation for CSV handling (at https://reference.wolfram.com/language/ref/format/CSV.html). Note in particular the possibility of specifying a `DataStringFormat`. Download US unemployment data in CSV format from the Federal Reserve Economic Database (e.g., https://research.stlouisfed.org/fred2/series/UNRATE/downloaddata). Plot this data using `DateListPlot`. You might end up with code resembling:

```
Import[datadir <> "UNRATE.csv",
  "DateStringFormat" → {"Year", "-", "Month", "-", "Day"}
 ];
DateListPlot[%[[2 ;;]]]
```



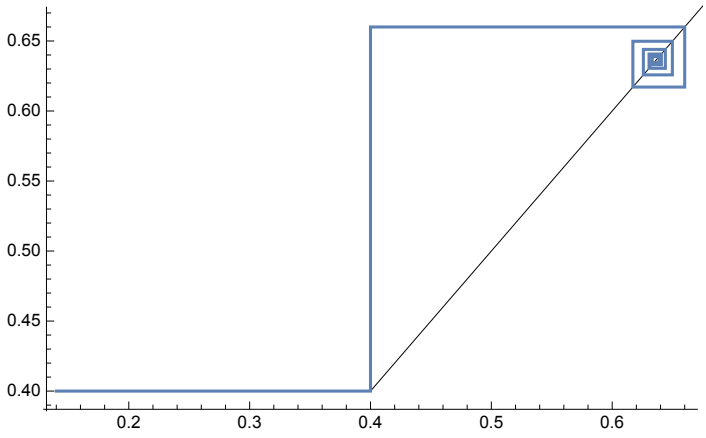
Cobweb Plot

Cobweb plots are a popular way to represent the evolution of first-order recurrence relationship.

As of *Mathematica* 10.2, we can use `ListStepPlot` to almost get a good cobweb diagram. Unfortunately, as of version 10.2, it is not yet possible to get rid of a line segment that is outside of the data.

```
logisticPath = NestList[2.75 * # * (1 - #) &, 0.4, 10]
ListStepPlot[Transpose[{logisticPath, logisticPath}], "Left", PlotRange → All,
 Prolog → Line[{{0.4, 0.4}, {0.8, 0.8}}]]
(* as of Mma 10.2 can use StepLinePlot *)
```

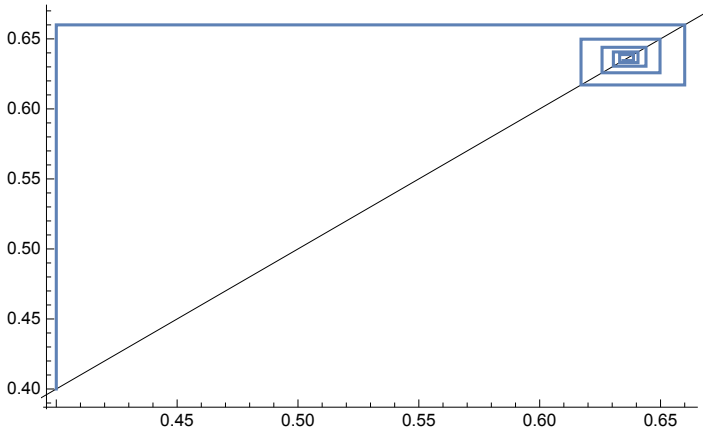
```
{0.4, 0.66, 0.6171, 0.649791, 0.625797,
 0.643981, 0.630491, 0.640673, 0.63308, 0.638797, 0.634523}
```



This limitation means that for a good cobweb plot, we are still forced to manipulate the data.

```
cobwebPoints = Partition[Riffle[logisticPath, logisticPath], 2, 1]
ListLinePlot[cobwebPoints, PlotRange → All,
 Prolog → {Line[{{0, 0}, {1, 1}}]},
 PlotRange → All]
```

```
{{0.4, 0.4}, {0.4, 0.66}, {0.66, 0.66}, {0.66, 0.6171}, {0.6171, 0.6171},
 {0.6171, 0.649791}, {0.649791, 0.649791}, {0.649791, 0.625797},
 {0.625797, 0.625797}, {0.625797, 0.643981}, {0.643981, 0.643981},
 {0.643981, 0.630491}, {0.630491, 0.630491}, {0.630491, 0.640673},
 {0.640673, 0.640673}, {0.640673, 0.63308}, {0.63308, 0.63308}, {0.63308, 0.638797},
 {0.638797, 0.638797}, {0.638797, 0.634523}, {0.634523, 0.634523}}
```



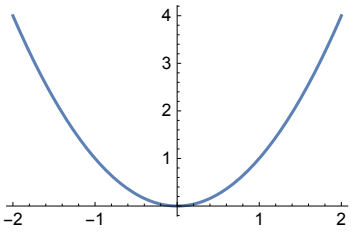
Exploring Functions with Plots

For two-dimensional function plotting, use the Plot command. For three-dimensional function plotting, use the Plot3D command. Get the *Mathematica* help in the usual way, and read it carefully.

Basic 2D Function Plotting

The Plot command provides simple 2D function plots. The first argument is an expression that involves a variable, and the second argument specifies the domain for that variable. The second argument is in the form {x,xmin,xmax}, and such *Mathematica* calls such expressions “iterators”.

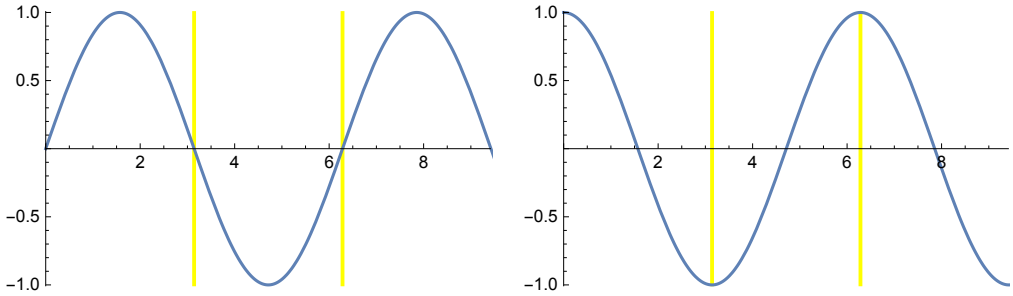
```
Clear[x]
Plot[x^2, {x, -2, 2}, ImageSize → Small]
```



Fine control can be achieved with the many options, which you can examine by giving the command `Options[Plot]`. Naturally, *Mathematica* provides help for each option. If you have a style you would like

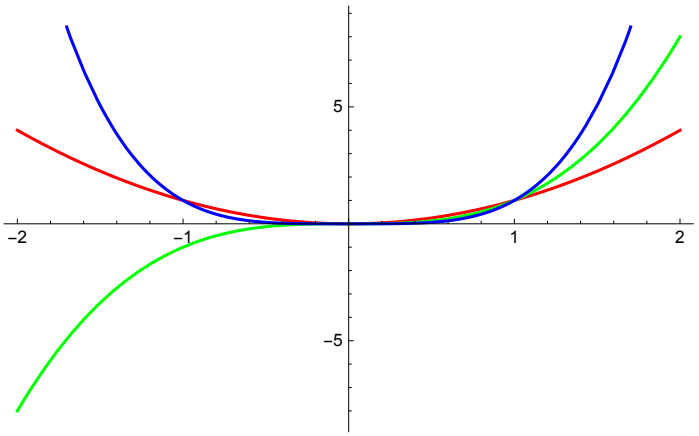
to apply to multiple plots, you can collect the options in a list, say `myoptions`, and then using `Evaluate[myoptions]` when you need it. In our notebooks, we often use the `ImageSize` option, as above. You can also use the `ImageSize` option to specify the width or height of your plot, in pixels.

```
myoptions = {ImageSize → 250, PlotRange → {{0, 3 * Pi}, {-1.01, 1.01}},
    GridLines → {{Pi, 2 Pi}, {}}, GridLinesStyle → Directive[Thick, Yellow]};
GraphicsRow[{Plot[Sin[x], {x, 0, 10}, Evaluate[myoptions]],
  Plot[Cos[x], {x, 0, 10}, Evaluate[myoptions]]}]
```



You can specify a list of expressions to be plotted together. You can also specify colors with the `PlotStyle` option.

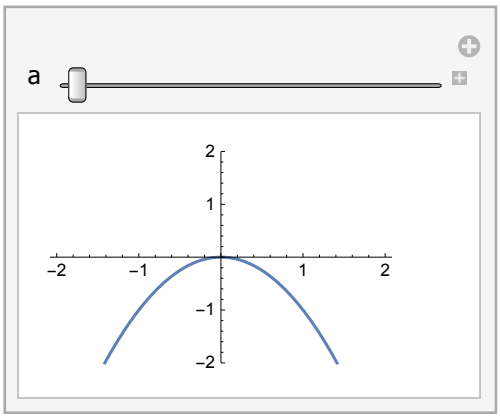
```
Clear[x]
Plot[{x^2, x^3, x^4}, {x, -2, 2},
 ImageSize → Medium, PlotStyle → {Red, Green, Blue}]
```



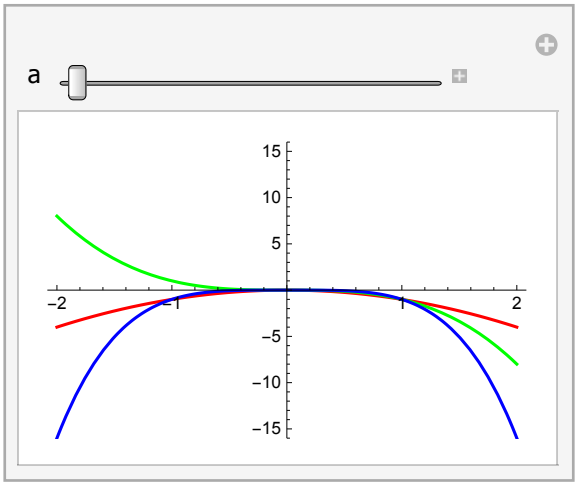
Advanced: Manipulate

Use Manipulate to allow dynamic exploration of function plots.

```
Clear[a, x]
Manipulate[Plot[a * x^2, {x, -2, 2}, PlotRange → {-2, 2}, ImageSize → Small],
 {a, -1, 1}]
```




```
Clear[a, x]
Manipulate[
 Plot[{a * x^2, a * x^3, a * x^4}, {x, -2, 2}, PlotRange → {-16, 16},
  ImageSize → 250, PlotStyle → {Red, Green, Blue}],
 {a, -1, 1}]
```

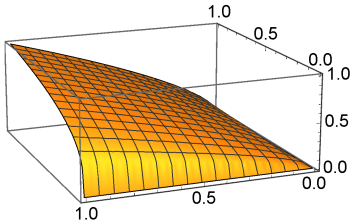


Basic 3D Function Plotting

The Plot3D command provides simple 3D surface plots of functions. The first argument is an expression that involves two variables, and the second and third arguments specify the domain for those variable. The second and third arguments are in the form {x,xmin,xmax}, and such *Mathematica* calls such expressions “iterators”. We can specify additional options, such as ImageSize, order to adjust the plot characteristics. The most important of may be `ViewPoint`, which sets the point in space from which the plot is viewed. For advanced usage, see http://www.wolfram.com/training/courses/vis422.html

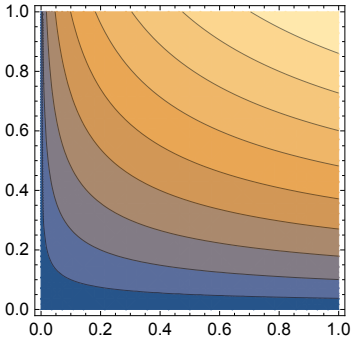
```
Clear[k, n]
cd = k^α * n^(1 - α) /. {α → 0.3}
Plot3D[cd, {k, 0, 1}, {n, 0, 1}, ImageSize → Small, ViewPoint → {-5, 2, 1.5}]
```

$k^{0.3}\, n^{0.7}$

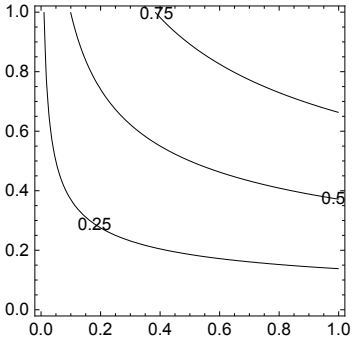


Sometimes it is convenient to view three dimensional functions as contours.

```
ContourPlot[cd, {k, 0, 1}, {n, 0, 1}, ImageSize → Small]
```




```
ContourPlot[cd, {k, 0, 1}, {n, 0, 1},
 Contours → {0.25, 0.5, 0.75},
 ContourLabels → All,
 ContourShading → None,
 ImageSize → Small]
```



Basics: Sets, Vectors, and Matrices

Lists as Sets

Mathematica does not have a separate set type (as of version 10). Instead, it defines set operations on lists.

Lists differ from sets in that order matters and duplicate elements matter. We can explicitly discard duplicate elements with the `DeleteDuplicates` command.

```
s1 = {1, 2, 2, 3, 3, 3};
DeleteDuplicates[s1]
```

```
{1, 2, 3}
```

Basic Set Operations and Set Relations

For more detail than provided in this section, see the Wolfram Language Guide entitled Operations on Sets.

Union and Intersection

The set operation commands `Union` and `Intersection` conveniently discard duplicates from each set.

```
s1 = {1, 2, 2, 3, 3, 3}; s2 = {3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5};
Union[s1, s2]
Intersection[s1, s2]
```

```
{1, 2, 3, 4, 5}
```

```
{3}
```

However, the Subsets command does not discard duplicates.

```
Subsets[{1, 1, 2}]
```

```
{{}, {1}, {1}, {2}, {1, 1}, {1, 2}, {1, 2}, {1, 1, 2}}
```

Set Difference (Complement)

Mathematica offers the `Complement` command to perform set difference:

```
s1 = {1, 2, 2, 3, 3, 3}; s2 = {3, 4, 5};
Complement[s1, s2]
```

```
{1, 2}
```

Mathematica does not currently offer a `SymmetricDifference` command, so for this you need to use the union of the set differences.

```
s1 = {1, 1, 2, 3}; s2 = {3, 4, 5};
Union[Complement[s1, s2], Complement[s2, s1]]
```

```
{1, 2, 4, 5}
```

Elements and Subsets

Mathematica (v.10) offers the `MemberQ` command to test elementhood and the `SubsetQ` command to test inclusion. Note the perhaps surprising order of arguments: here we test if s2 ⊆ s1.

```
s1 = Range[10]; s2 = {1, 5};
MemberQ[s1, 5]
SubsetQ[s1, s2]
```

```
True
```

```
True
```

We can use `Map` to perform many tests at one go. (Note that object type matters for membership tests.)

```
Map[x ↦ MemberQ[s1, x], {0, 1, 1.0}]
Map[x ↦ SubsetQ[s1, x], {{0}, {1}, {9.0}, {10}}]
```

```
{False, True, False}
```

```
{False, True, False, True}
```

Let's build our own subset text, using the `MemberQ` command for testing elementhood. Working directly with the definition of subset, we might use `Apply`, `And`, and `Map` to come up with the following test of whether `set2` is a subset of `set1`. (See the documentation of `Apply`. Note that we match the *Mathematica* argument order.)

```
ClearAll[set1, set2, subsetQ01]
subsetQ01 = Function[{set1, set2},
  Apply[And, Map[x ↦ MemberQ[set1, x], set2]]
 ]
subsetQ01[Range[10], {1, 5}]
subsetQ01[{1, 5}, Range[10]]
```

```
Function[{set1, set2}, And @@ Function[x, MemberQ[set1, x]] /@ set2]
```

```
True
```

```
False
```

If you review the output of our function definition, note that it uses the `/@` shorthand notation for `Map` and the `@@` shorthand notation for `Apply`.

Exercise: Describe the action of the following anonymous function. Assume the inputs are two lists.

```
And @@ ((x ↦ MemberQ[#1, x]) /@ #2) &
```

```
And @@ Function[x, MemberQ[#1, x]] /@ #2 &
```

Our work so far provides a perfectly good solution. But rather than relying so directly on the definition of subset, it turns out to be much faster to use an implication of the defintion. (It could be even faster if we could be sure ahead of time that there are no duplicates in the proposed subset.)

```
Clear[set1, set2, subsetQ]
(*
subsetQ[set1_List,set2_List]:=Module[{s2},
  (* return bool, True if set2 is a subset of set1 *)
  s2=DeleteDuplicates[set2];
  Length[Intersection[set1,s2]]==Length[s2]
 ]
*)
subsetQ = {set1, set2} ↦ Module[{s2 = DeleteDuplicates[set2]},
   Length[Intersection[set1, s2]] == Length[s2]
  ]
Function[{set1, set2},
 Module[{s2 = DeleteDuplicates[set2]}, Length[set1⋂s2] == Length[s2]]]
```

We can use the `Timing` command to compare the computational speed of the two approaches.

```
s1 = Range[10 000];  s2 = RandomSample[s1, 1000];
Timing[subsetQ01[s1, s2]]
Timing[subsetQ[s1, s2]]
```

```
{0.577204, True}
```

```
{0., True}
```

Set Building

Criterion-based filtering is a basic need in data manipulation. Use the `Select` command to construct smaller sets from larger sets based on an inclusion criterion.

```
lst = Range[10]
Select[lst, EvenQ]
Select[lst, x ↦ 3 < x < 8]
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

```
{2, 4, 6, 8, 10}
```

```
{4, 5, 6, 7}
```

For example, we might retrieve data from a lifetable, and be interested only in the data for children under the age of 12.

```
data = ExampleData[{"Statistics", "USLifeTable2003"}];
data12 = Select[data, x ↦ x[[1]] < 12];
```

Permutations and Combinations

Permutations

```
Permutations[Range[3]] (* all permuations *)
Permutations[Range[5], {3}] (* all permutations of length 3 *)
```

```
{{1, 2, 3}, {1, 3, 2}, {2, 1, 3}, {2, 3, 1}, {3, 1, 2}, {3, 2, 1}}
```

```
{{1, 2, 3}, {1, 2, 4}, {1, 2, 5}, {1, 3, 2}, {1, 3, 4}, {1, 3, 5}, {1, 4, 2}, {1, 4, 3},
 {1, 4, 5}, {1, 5, 2}, {1, 5, 3}, {1, 5, 4}, {2, 1, 3}, {2, 1, 4}, {2, 1, 5}, {2, 3, 1},
 {2, 3, 4}, {2, 3, 5}, {2, 4, 1}, {2, 4, 3}, {2, 4, 5}, {2, 5, 1}, {2, 5, 3}, {2, 5, 4},
 {3, 1, 2}, {3, 1, 4}, {3, 1, 5}, {3, 2, 1}, {3, 2, 4}, {3, 2, 5}, {3, 4, 1}, {3, 4, 2},
 {3, 4, 5}, {3, 5, 1}, {3, 5, 2}, {3, 5, 4}, {4, 1, 2}, {4, 1, 3}, {4, 1, 5},
 {4, 2, 1}, {4, 2, 3}, {4, 2, 5}, {4, 3, 1}, {4, 3, 2}, {4, 3, 5}, {4, 5, 1},
 {4, 5, 2}, {4, 5, 3}, {5, 1, 2}, {5, 1, 3}, {5, 1, 4}, {5, 2, 1}, {5, 2, 3},
 {5, 2, 4}, {5, 3, 1}, {5, 3, 2}, {5, 3, 4}, {5, 4, 1}, {5, 4, 2}, {5, 4, 3}}
```

The Permutations command will use repeated elements, but they are treated as identical.

```
Permutations[{x, x, y}]
```

```
{{x, x, y}, {x, y, x}, {y, x, x}}
```

Combinations

We can use the Subsets command with a second argument of {k} (note the braces) to produce k-subsets (combinations):

```
Subsets[{1, 2, 3}, {2}]
```

```
{{1, 2}, {1, 3}, {2, 3}}
```

Pascal's Triangle

Exemplifying Stigler's law of eponymy, the triangular arrangement of the binomial coefficients known as Pascal's triangle was developed by the 10th century Persian mathematician Al-Karaji.

```
pascal = Table[Binomial[n, k], {n, 0, 10}, {k, 0, n}]
Grid[pascal, Alignment → Right]
```

```
{{1}, {1, 1}, {1, 2, 1}, {1, 3, 3, 1}, {1, 4, 6, 4, 1}, {1, 5, 10, 10, 5, 1},
 {1, 6, 15, 20, 15, 6, 1}, {1, 7, 21, 35, 35, 21, 7, 1}, {1, 8, 28, 56, 70, 56, 28, 8, 1},
 {1, 9, 36, 84, 126, 126, 84, 36, 9, 1}, {1, 10, 45, 120, 210, 252, 210, 120, 45, 10, 1}}
```

```
1
1   1
1   2   1
1   3   3    1
1   4   6    4    1
1   5  10   10    5    1
1   6  15   20   15    6    1
1   7  21   35   35   21    7    1
1   8  28   56   70   56   28    8   1
1   9  36   84  126  126   84   36   9   1
1  10  45  120  210  252  210  120  45  10  1
```

We can produced the same triangle with a bit better formatting.

```
maxitemsize = IntegerLength@Max@Last@pascal
Grid[pascal, Alignment → Right, ItemSize → maxitemsize - 1, Spacings → 0]
```

```
3
```

```
   1
   1   1
   1   2   1
   1   3   3   1
   1   4   6   4   1
   1   5  10  10   5   1
   1   6  15  20  15   6   1
   1   7  21  35  35  21   7   1
   1   8  28  56  70  56  28   8   1
   1   9  36  84 126 126  84  36   9   1
   1  10  45 120 210 252 210 120  45  10   1
```

We can produced the same triangle with a more traditional formatting.

```
pascalTri6[n_] := Module[{max, cellWd}, (*Maximum entry. Cell width.*)
  max = Max[Table[Binomial[n, j], {j, 0, n}]];
  cellWd = 0.25 * IntegerLength[max] + 2;
  Column[Table[Grid[{Table[Binomial[i, j], {j, 0, i}]},
     ItemSize → {cellWd, 2}, Alignment → Center], {i, 0, n}], Center]]
pascalTri6[
 10]
```

```
                                1
                              1   1
                            1   2   1
                          1   3   3   1
                        1   4   6   4   1
                      1   5   10  10  5   1
                    1   6   15  20  15  6   1
                  1   7   21  35  35  21  7   1
                1   8   28  56  70  56  28  8   1
              1   9   36  84  126 126 84  36  9   1
            1   10  45  120 210 252 210 120 45  10  1
```

Lists as Vectors

Mathematica lists support elementwise addition and scalar multiplication. Because of this, lists with *N* numerical elements give us a natural computational representation of *N*-vectors.

Introduction to Vectors

In this booklet we will be mostly interested in real, finite-dimensional vectors.

Creating and Indexing Vectors

Any list of numbers can represent a vector, so all the tools we developed for creating lists are applicable to creating vectors. Here are some two-dimensional examples. (We end each statement with a semi-colon, which suppresses the output.)

```
v0 = {0, 0}; v1 = {1, 2}; v2 = {2, 1};
```

Notice how we can refer to each list with a single symbol. Notice that order matters: v_1 and v_2 represent two different vectors.

Recall that *Mathematica* uses unit-based indexing: the first coordinate has an index of 1. Indexing uses double brackets.

```
v1 = {1, 2}; v1[[1]]
```

```
1
```

A vector is an element of a vector space. A vector space is essentially a set whose elements (vectors) are closed under scalar multiplication and addition. In *Mathematica*, scalar multiplication and vector addition work naturally.

```
2 * v1 (* double each element of v1 *)
v0 + v1 (* add corresponding element of v1 and v2 *)
2 * v1 + 3 * v2 (* linear combination of v1 and v2 *)
```

```
{2, 4}
```

```
{1, 2}
```

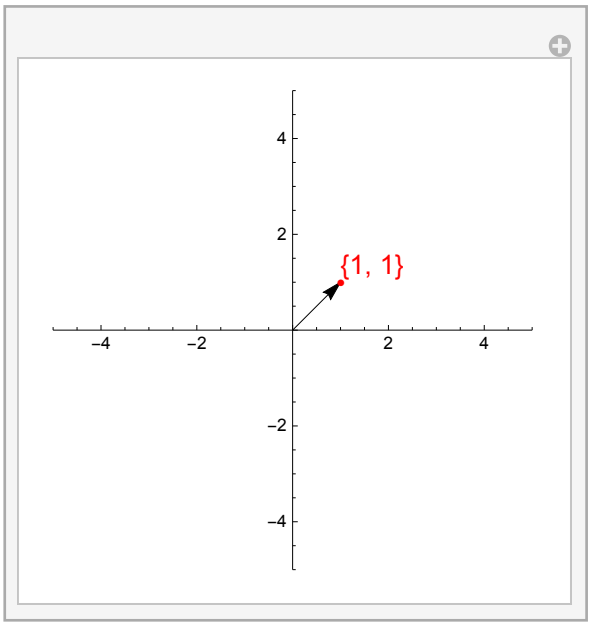
```
{8, 7}
```

Visual Representation of Vectors in 2-Space

In the Cartesian coordinate system, each coordinate represents a distance along a coordinate axis. This leads to two standard graphical representations of 2-tuples and 3-tuples: as simple filled discs, or as the end of an arrow whose tail is at the origin. The second representation is often used to indicate that the tuples are vectors.

Two-dimensional real vectors have a familiar representation with standard Cartesian coordinates.

nopdf, noclass



Visual Representation of Vectors in 3-Space

noclass, nopdf

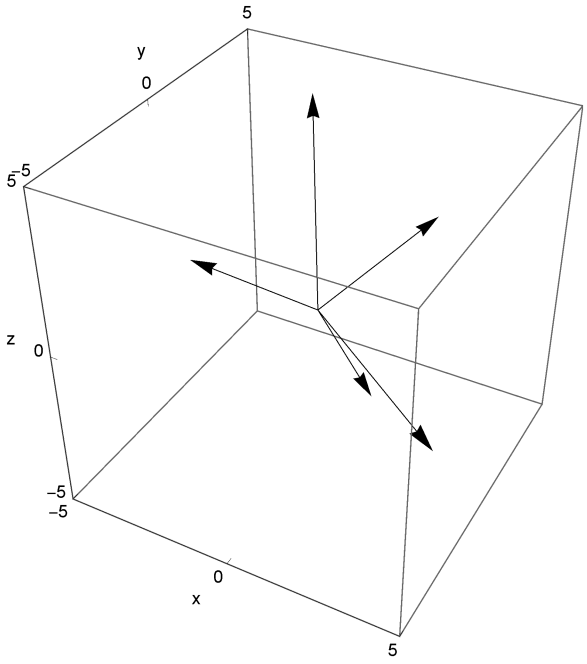


Illustration: Scalar Multiplication

Scalar multiplication literally scales the vector.

```
Clear[s, x1, x2]
s * {x1, x2}
```

```
{s x1, s x2}
```

We can easily visualize this in two dimensions:

noclass, nopdf

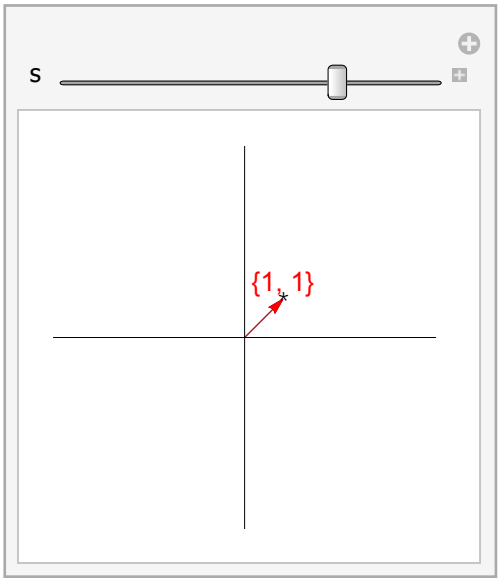


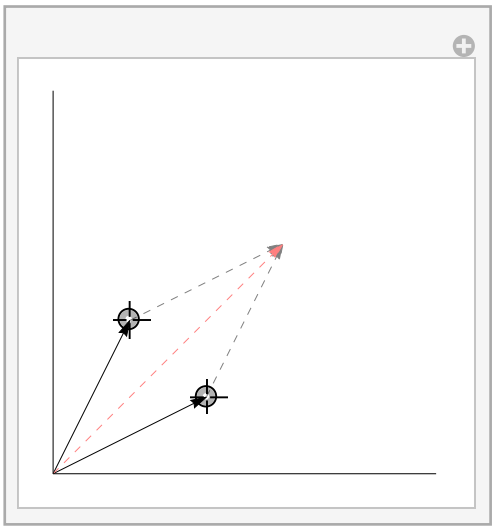
Illustration: Vector Addition

```
Clear[x1, x2, y1, y2]
{x1, x2} + {y1, y2}
```

```
{x1 + y1, x2 + y2}
```

Scalar addition can be represented as “completing a parallelogram”.

noclass, nopdf



Scalar Product

The scalar product, or dot product can be broken into two steps: elementwise multiplication, and summa-

tion of elements.

Since *Mathematica* does not distinguish between lists and vectors, all list operations are available if you wish to do non-standard vector manipulations. For example, you can perform element by element muliplication simply by using the multiplcation operator (which is equivalent to the Times command). For two conformable vectors, this produces the element-wise product (which is also called the Hadamard product).

```
v1 = {1, 2}; v2 = {3, 4};
v3 = v1 * v2
```

```
{3, 8}
```

Some times we want to produce a scalar result by operating on a vector. For example, we may want the sum of the elements. *Mathematica* provides the Total command for this.

```
v3 = {3, 8};
Total[v3]
```

```
11
```

Suppose we first do element-by-element multiplication of two vectors and then sum the elements of the result: this produces what is known as the “scalar product” (or “dot product”) of our two vectors. The scalar product of two vectors in $\mathbb{R}^n$ is the result of multiplying the corresponding coordinates and summing those products.

```
v1 = {a, b}; v2 = {c, d};
Total[v1 * v2]
```

```
a c + b d
```

Since we have discussed how to use `Apply` and `Plus` to produce this total, we can also use `Apply` to produce and ordinary dot product of vectors:

```
Apply[Plus, v1 * v2]
```

```
a c + b d
```

As a short-cut, there is also an infix operator for Apply: @@.

```
Plus @@ (v1 * v2)
```

```
a c + b d
```

The “dot product” is such a common need that *Mathematica* provides the Dot command to more simply produce this result. As a convenient shorthand, *Mathematica* provides the period as an infix operator.

```
Dot[v1, v2]
% === v1.v2
```

```
a c + b d
```

```
True
```

Advanced: Inner Product

For vectors in $\mathbb{R}^n$, the dot product is an inner product. An inner product of two vectors is often represented as < v1, v2 >. For real vectors, you can use *Mathematica*'s Inner command to produce the dot product.

```
Clear[a, b, c, d]
Inner[Times, {a, b}, {c, d}, Plus]
```

```
a c + b d
```

An inner product maps two vectors to a number while meeting certain constraints, which we will discuss in more detail latter. For vectors of real numbers one of these constraints is that < v1, v2 > == < v2, v1 >. Note that *Mathematica* s's Inner command does not enforce such constraints. For example, let us replace Times with Power.

```
Clear[a, b, c, d]
Inner[Power, {a, b}, {c, d}, Plus]
Inner[Power, {c, d}, {a, b}, Plus]
```

```
a^c + b^d
```

```
c^a + d^b
```

Linear Combination

A finite weighted sum of vectors is called a linear combination of the vectors. If the weights sum to unity, it is also called an affine combination. An affine combination with nonnegative weights is called a convex combination.

Examples

Here we create two 2-vectors and then scale them, purely symbolically.

```
Clear[s1, s2, v11, v12, v21, v22]
v1 = {v11, v12}; v2 = {v21, v22}
s1 * v1 + s2 * v2  (* linear combination of v1 and v2 *)
```

```
{v21, v22}
```

```
{s1 v11 + s2 v21, s1 v12 + s2 v22}
```

Here is a similar exercise, but using numbers instead of symbols.

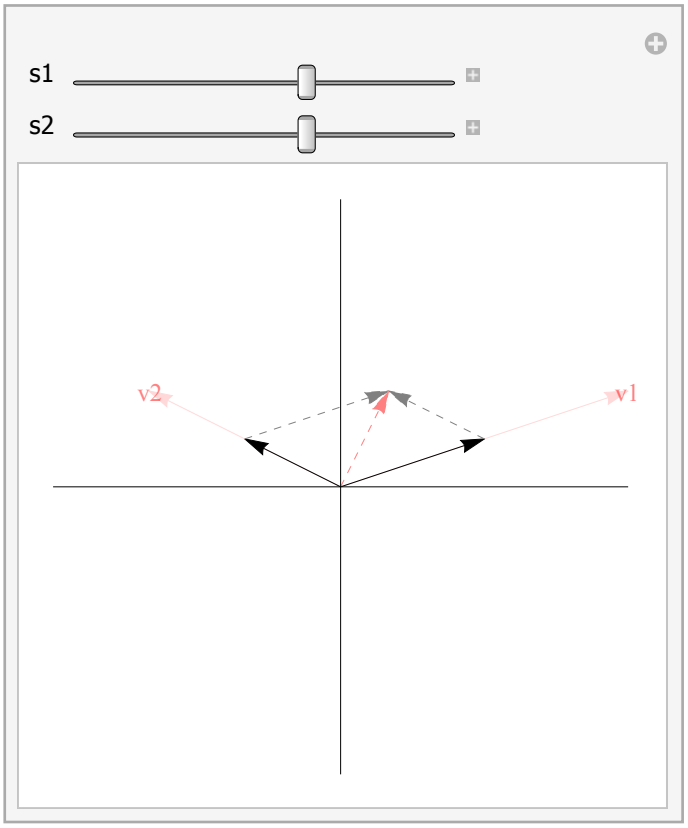
```
v1 = {1, 2}; v2 = {3, 4}; v3 = {5, 6};
1 * v1 + 2 * v2 + 3 * v3 (* linear combination of v1, v2, and v3 *)
```

```
{22, 28}
```

Illustration: Linear Combination

In the following illustration, we produced a linear combination (i.e., weighted sum) of two vectors.

noclass, nopdf



Vector Space, Span, and Linear Independence

A vector space is a set of vectors that is closed under linear combination. This means that any linear combination of vectors in the set produces another vector in the set.

xmpl

Example

Consider the following set of vectors: any vector x, along with all the vectors that can be formed from it by scalar multiplication. If we add any two vectors from this set, we get another vector in this set.

Let V be a collection of vectors. The linear span of V is the set of all the vectors that can formed as linear combinations of vectors in V. The span of a collection of vectors is clearly a vector space.

Any vector in the span of V is said to be linearly dependent on V. Any vector not in the span of V is said to be linearly indepenendent of V.

xmpl

Example

Let $V = \{v_1, v_2\}$ where $v_1 = (1, 2)$ and $v_2 = (2, 4)$. Then $v_3 = (7, 10)$ is linearly independent of V: we cannot find a linear combination of v_1 and v_2 that is equal to v_3. That is, we cannot find two scalars (s_1, s_2) such that $v_3 = s_1 v_1 + s_2 v_2$.

We can still ask *Mathematica* to try to find solutions when none exist, but we will get an empty set of solutions. For example, let us attempt to use `Solve` on the following equation.

```
Solve[s1 * {1, 2} + s2 * {2, 4} == {7, 10}, {s1, s2}]
```

```
{}
```

In contrast, $v_4 = (7, 14)$ is linearly dependent on V. For example, $v_4 = 3\,v_1 + 2\,v_2$. Let us try to show this using `Solve`.

```
Solve[s1 * {1, 2} + s2 * {2, 4} == {7, 14}, {s1, s2}]
```

```
Solve::svars : Equations may not give solutions for all "solve" variables. >>
```

$$\left\{\left\{s2 \to \frac{7}{2} - \frac{s1}{2}\right\}\right\}$$

What happened? A collection of vectors V is said to be linearly independent if none of the vectors lies in the span of the others. The problem is that V is itself linearly dependent, so there is not a unique solution to this problem. *Mathematica* gave us an answer that represents all the solutions, including our proposed solution $(s_1, s_2) = (3, 2)$.

Here are some ways to test for linear independence.

```
Solve[s1 * {1, 2} + s2 * {3, 4} == {7, 10}, {s1, s2}]
(* (7,10) is in the span of (1,2) and (3,4) *)
```

```
{{s1 → 1, s2 → 2}}
```

Linear Equations

Null Space

The null space of a vector a is the set of vectors that are transformed to 0 by a: null(a) = {x | $a.x = 0$}.

The *Mathematica* command `NullSpace` takes a matrix argument and returns a basis for the nullspace of that matrix.

```
mA = Transpose[{{1, 2}, {3, 4}}]; (* put our vectors in the columns *)
nA = NullSpace[mA]
```

```
{}
```

```
v1 = {1, 2}; v2 = {3, 6};
mA = Transpose[{v1, v2}]; (* put our vectors in the columns *)
nA = NullSpace[mA]
```

```
{{-3, 1}}
```

This gives us weights for a linear combination of our vectors that equals the zero vector.

```
-3 * v1 + 1 * v2
```

```
{0, 0}
```

```
mA.Transpose[nA]
```

```
{{0}, {0}}
```

Null Space

Suppose we want to characterize the null space of the vector (1,2). One approach is to use Solve to find the equation of the null space:

```
Clear[x1, x2]
v1 = {1, 2};
Solve[v1.{x1, x2} == 0, {x1, x2}]
```

```
Solve::svars : Equations may not give solutions for all "solve" variables. >>
```

$$\left\{\left\{x2 \to -\frac{x1}{2}\right\}\right\}$$

Another approach is to use *Mathematica*'s NullSpace command, which returns a basis for the null space. However, this only works on matrices, so we would have to wrap the vector in a list:

```
NullSpace[{v1}]
```

```
{{-2, 1}}
```

```
m1 = {{1}, {2}}
m1 // MatrixForm
NullSpace[m1]
```

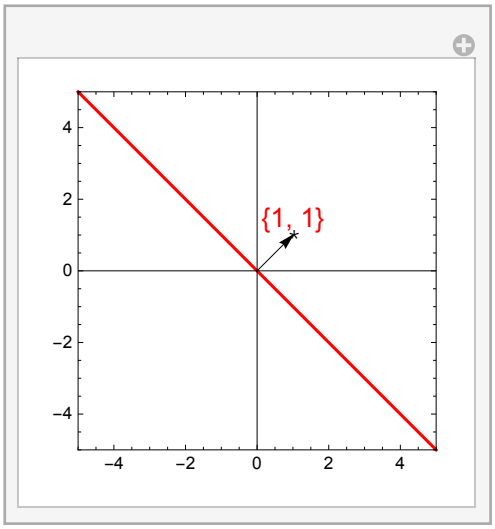
```
{{1}, {2}}
```

$$\begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

```
{}
```

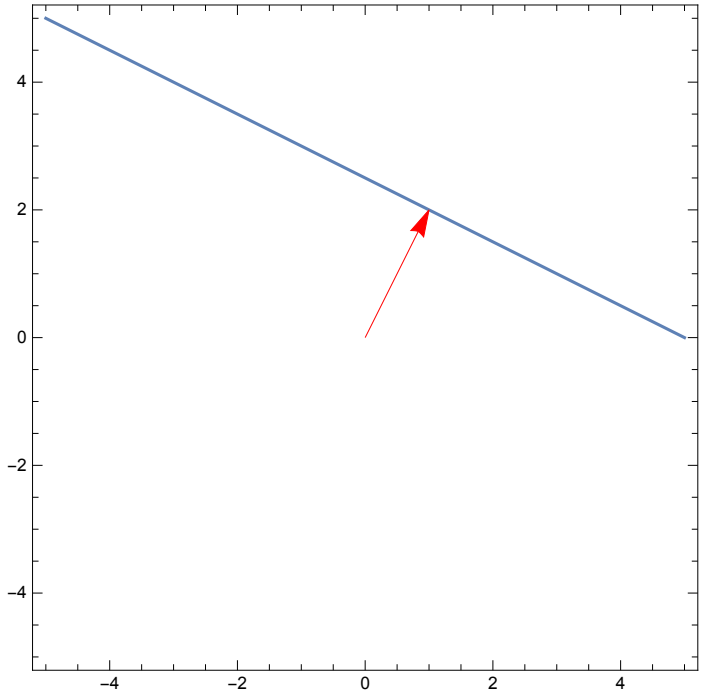
Illustrate Null Space: 2D

```
swf = Manipulate[ContourPlot[v.{x1, x2} == 0, {x1, -5, 5}, {x2, -5, 5},
    PlotRange → {{-5, 5}, {-5, 5}},
    ImageSize → {200, 200},
    Axes → True, Ticks → False,
    ContourStyle → Red,
    Epilog → {
      Arrow[{{0, 0}, v}],
      {Red, Style[Text[Style[ToString[v]], v, {0, -Sign[v[[2]]]}], Larger]}
     }
  ],
  {{v, {1, 1}}, Locator, Appearance → "*"}
 ]
```



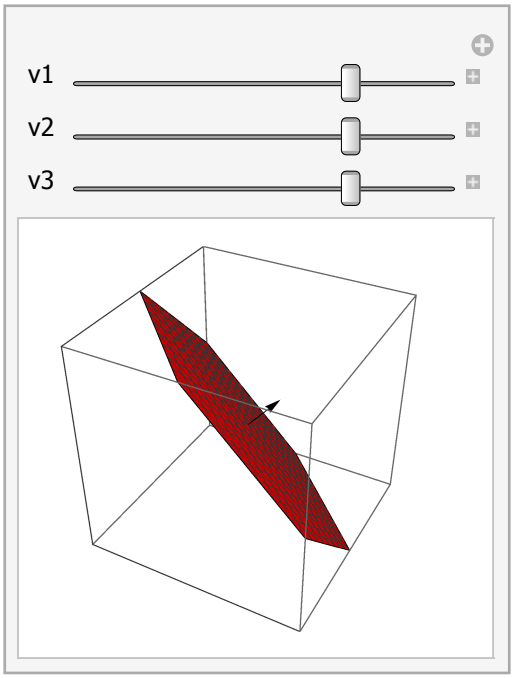
```
v = {1, 2}
ContourPlot[v.{x1, x2} == 5, {x1, -5, 5},
 {x2, -5, 5}, Epilog → {Red, Arrow[{{0, 0}, v}]}]
```

```
{1, 2}
```



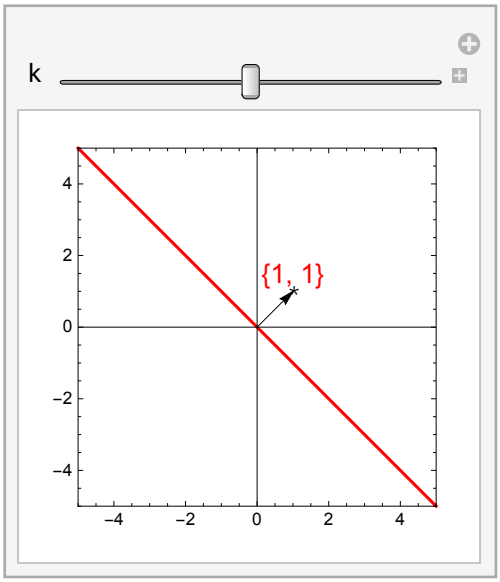
Illustrate Nullspace: 3D

```
anim = Manipulate[
  Show[{
    ContourPlot3D[{v1, v2, v3}.{x1, x2, x3} == 0, {x1, -5, 5}, {x2, -5, 5}, {x3, -5, 5},
     PlotRange → {{-5, 5}, {-5, 5}, {-5, 5}},
     ImageSize → {200, 200},
     Axes → True, Ticks → False,
     ContourStyle → Red],
    Graphics3D[Arrow[{{0, 0, 0}, {v1, v2, v3}}]]}
  ],
  {{v1, 1}, -2, 2}, {{v2, 1}, -2, 2}, {{v3, 1}, -2, 2}
 ]
(* Export["c:\\temp\\temp.avi",anim,"ControlAppearance"→Automatic] *)
```



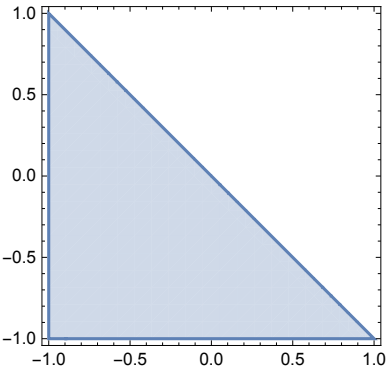
Affine Spaces: Illustrating Hyperplanes

```
Manipulate[ContourPlot[v.{x1, x2} == k, {x1, -5, 5}, {x2, -5, 5},
  PlotRange → {{-5, 5}, {-5, 5}},
  ImageSize → {200, 200},
  Axes → True, Ticks → False,
  ContourStyle → Red,
  Epilog → {
    Arrow[{{0, 0}, v}],
    {Red, Style[Text[Style[ToString[v]], v, {0, -Sign[v[[2]]]}], Larger]}
   }
 ],
 {{k, 0}, -5, 5}, {{v, {1, 1}}, Locator, Appearance → "*"}
]
```



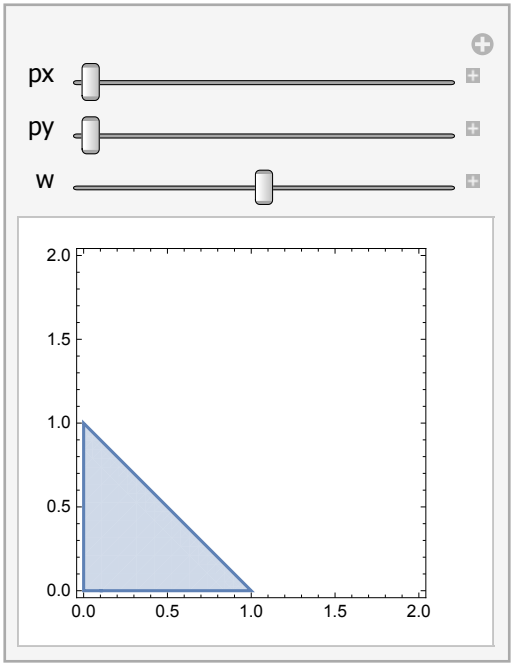
Inequalities

```
RegionPlot[x + y ≤ 0, {x, -1, 1}, {y, -1, 1}, ImageSize → 200]
```



Budget Constraints

```
Manipulate[RegionPlot[px * x + py * y ≤ w, {x, 0, 2}, {y, 0, 2}, ImageSize → 200],
 {px, 1, 5}, {py, 1, 5}, {{w, 1}, 0, 2}]
```



Linear Functions

Lines

Consider any two distinct points, p_1 and p_2. The line segment from p_1 to p_2 is the shortest path between the two points, and the length of this path is the distance between the two points. This line segment is the set of points that can be produced as convex combinations of p1 and p2. The line through p1 and p2 is the set of points that can be produced as affine combinations of the two points. That is, any point p on the line can be expressed as

$$p = (1 - t)\, p_1 + t\, p_2 = p_1 + t\, (p_2 - p_1)$$

This is called the parametric form of the line. We will refer to the point $(p_2 - p_1)$ as a "direction vector" for the line. To illustrate this, suppose our two distinct points are in the Cartesian plane: $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$. Then we have

$$x = x_1 + t\, (x_2 - x_1)$$

$$y = y_1 + t\, (y_2 - y_1)$$

which implies

$$(y_2 - y_1)\, (x - x_1) = (x_2 - x_1)\, (y - y_1)$$

To see this, note that we can rewrite our system as

$$(y_2 - y_1)\, (x - x_1) = (y_2 - y_1)\, (x_2 - x_1)\, t$$

$$(x_2 - x_1)\,(y - y_1) = (x_2 - x_1)\,(y_2 - y_1)\,t$$

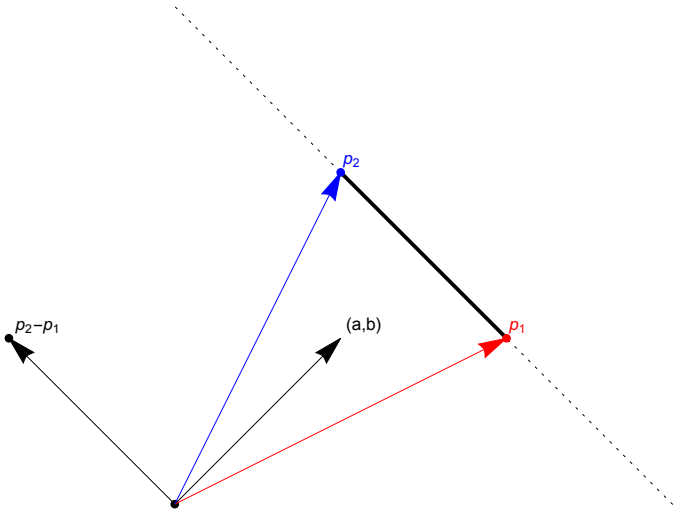
This line is commonly represented by the equation

$$a\,x + b\,y + c = 0$$

where e.g. $a = y_2 - y_1$, $b = x_1 - x_2$, and $c = y_1\,(x_2 - x_1) - x_1(y_2 - y_1) = y_1\,x_2 - x_1\,y_2$.

```
p1 = {x1, y1} = {2, 1}; p2 = {x2, y2} = {1, 2};
(* arbitrary coordinates for two points *)
a = y2 - y1; b = x1 - x2; (* define a and b *)
```

nopdf, noclass



If $x_1 \neq x_2$ we define the slope m of the line by

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

Otherwise the slope is undefined. Two distinct lines in a plane are “parallel” if they do not intersect; the minimum distance from one to the other is constant. Parallel lines have a common slope. Two parallel lines cut the horizontal axis at the same angle, denoted by θ. The slope and the angle are related by

$$m = \tan(\theta)$$

$$\theta = \arctan(m)$$

$$\mathbf{m = \frac{y2 - y1}{x2 - x1}}$$

```
ArcTan[-1, 1]
```

$$-1$$

$$\frac{3\,\pi}{4}$$

Two lines in a plane are “perpendicular” if their intersection creates common adjacent angles (i.e., right angles). The product of their slopes is −1 (if defined). The dot product of their direction vectors is 0.

Lists of Lists as Matrices

We create matrices as rectangular lists of lists.

Matrix Basics

Matrix Form

```
lst01 = {1, 2, 3}; lst02 = {2, 3, 4};
mA = {lst01, lst02}
```

```
{{1, 2, 3}, {2, 3, 4}}
```

These matrices can contain symbols as well as numbers.

```
Clear[a, b, c, d]
mB = {{a, b}, {c, d}}
```

```
{{a, b}, {c, d}}
```

If you wish to see your matrix displayed in a more tranditional way, you can use the MatrixForm command. (Be careful, however, as the value returned by this command is not a matrix and will not respond correctly to matrix operations.)

```
MatrixForm[mA]
MatrixForm[mB]
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \end{pmatrix}$$

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

Creating Matrices by Hand

This section will focus on matrix creation and manipulation. Remember that *Mathematica* uses capital letters for built-in commands, and some of these (C,D,E,I,K,N, and O) are single letter commands. So although traditional textbook notation uses capital letters for matrices, this is a risky practice for *Mathematica* input. By convention, user defined symbols begin with a lower-case letter.

Since a *Mathematica* matrix is just a rectangular list of lists, it is easy to create small matrices directly. These matrices can contain symbols as well as numbers.

If you wish to see your matrix displayed in a more traditional way, you can use the MatrixForm command. (Be careful to use this just for display, however. The value returned by this command is a display object, not a matrix, and may not respond correctly to matrix operations.)

```
Clear[a, b, c, d]
mB = {{a, b}, {c, d}}
MatrixForm[mB]
```

```
{{a, b}, {c, d}}
```

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

Convenient Matrix Entry

You can enter expressions in a two-dimensional layout by using Ctrl+, to add columns and Ctrl+Enter to add rows. By default, this produces a list of lists of placeholders. The placeholders allow for convenient keyboard entry of matrix elements, since you can use Tab to move from placeholder to placeholder.

Another easy way to type matrices is to pick the Basic Math Assistant palette, scroll down to the first typesetting tab, and click the matrix template. (Note that by default matrices display rounded brackets.) This will give you a 2 by 2 matrix template. Again, use Ctrl+Enter to add a row; use Ctrl+, to add a column. Here is a matrix created in this way: $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$.

If you are using the Basic Math Assistant, it is probably because you have forgotten the keyboard shortcuts. Hover your mouse over the template in the Math Assistant to see the keyboard shortcuts for matrix creation.

Indexing into Matrices

A matrix is just a rectangular list of lists. From a multidimensional list, we can extract a part of a part in the obvious ways (e.g., sequential double brackets). But as a convenient shorthand, we can just use a command separated list of indexes.

```
mA = Table[10 * (i - 1) + j, {i, 3}, {j, 5}] (* create a list of lists *)
mA[[2]][[2]] (* access the 2,2 element *)
mA[[2, 2]]  (* shorthand access of the 2,2 element *)
```

```
{{1, 2, 3, 4, 5}, {11, 12, 13, 14, 15}, {21, 22, 23, 24, 25}}
```

```
12
```

```
12
```

One can even arbitrarily select from rows or, more suprisingly, columns.

```
MatrixForm[mA]
mA[[All, 2]]  (* extract the second column *)
mA[[{1, 3}, 2]]  (* extract part of the second column *)
```

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 11 & 12 & 13 & 14 & 15 \\ 21 & 22 & 23 & 24 & 25 \end{pmatrix}$$

```
{2, 12, 22}
```

```
{2, 22}
```

Basic Vector Operations on Matrices

In *Mathematica*, a matrix is represented by a rectangular list of lists. The basic vector operations on matrices are very natural: add and subtract matrices with the usual + and - operators, and do scalar multiplication by premultiplying your matrix by any scalar.

```
Clear[mA, ones]; mA = {{1, 2}, {3, 4}}; ones = {{1, 1}, {1, 1}};
mA // MatrixForm
ones // MatrixForm
```

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

```
mA + ones // MatrixForm
mA - ones // MatrixForm
2 * ones // MatrixForm
```

$$\begin{pmatrix} 2 & 3 \\ 4 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}$$

$$\begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$$

This section will focus on matrix creation and manipulation. Remember that *Mathematica* uses capital letters for built-in commands, and some of these (C,D,E,I,K,N, and O) are single letter commands. So although traditional textbook notation uses capital letters for matrices, this is a risky practice for *Mathematica* input. By convention, user defined symbols begin with a lower-case letter.

Matrix Multiplication

The syntax for matrix multiplication may be a bit of a surprise for new users. *Mathematica* allows use of either a space or an asterisk to indicate element-by-element multiplication. Use a "dot" (i.e., a period or full-stop) to do matrix multiplication. Below we add comments, using *Mathematica*'s parentheses-with-asterisks comment notation, to emphasize this difference.

```
mA = {{1, 2}, {3, 4}}; ones = {{1, 1}, {1, 1}};
mA * ones  (* element-by-element multiplication (explicit Times) *)
mA ones   (* element-by-element multiplication (implicit Times) *)
mA . ones (* matrix multiplication (explicit Dot) *)
```

```
{{1, 2}, {3, 4}}
```

```
{{1, 2}, {3, 4}}
```

```
{{3, 3}, {7, 7}}
```

The infix operators * and . provide a shorthand for the *Mathematica* commands Times and Dot. In fact, *Mathematica* allows you to use these commands themselves as infix operators by surrounding them with tildes.

```
mA = {{1, 2}, {3, 4}}; ones = {{1, 1}, {1, 1}};
Times[mA, ones] === mA ~Times~ ones (* element-by-element multiplication *)
Dot[mA, ones] === mA ~Dot~ ones (* matrix multiplication *)
```

```
True
```

```
True
```

Transpose, Inverse, and Identity

Transpose and Inverse are naturally named. Note that built-in *Mathematica* functions are capitalized, and arguments to a *Mathematica* function must be placed in square brackets.

```
mA = {{a, b}, {c, d}}; mI = {{1, 0}, {0, 1}};
mI // MatrixForm
mA // MatrixForm
mI.mA // MatrixForm
mA.mI // MatrixForm
```

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

```
Map[IdentityMatrix, {1, 2, 3}]
Map[MatrixForm, %]
```

```
{{{1}}, {{1, 0}, {0, 1}}, {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}}}
```

$$\left\{(1), \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\right\}$$

```
mA = {{1, 2}, {3, 4}};
mAt = Transpose[mA]
mAI = Inverse[mA]
%.mA
```

```
{{1, 3}, {2, 4}}
```

$$\left\{\{-2, 1\}, \left\{\frac{3}{2}, -\frac{1}{2}\right\}\right\}$$

```
{{1, 0}, {0, 1}}
```

A symmetric matrix equals its transpose. If we matrix-multiply a matrix and its transpose, the result is symmetric.

```
mAAt = mAt.mA
mAAt == Transpose[mAAt]
```

```
{{10, 14}, {14, 20}}
```

```
True
```

An identity matrix is square, has ones on its diagonals, and all other elements are zero. Matrix-multiplication by an identity matrix has "no effect", in the sense that $A.I == I.A == A$. If we matrix-multiply a matrix A by its inverse A^{-1} we produce an identity matrix.

```
mI = IdentityMatrix[2]
mA.mI == mI.mA == mA
mA.mAI == mI
```

```
{{1, 0}, {0, 1}}
```

```
True
```

```
True
```

```
i3 = IdentityMatrix[3]; i3 // MatrixForm
mB = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}; mB // MatrixForm
i3.mB == mB
mB.i3 == mB
```

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

```
True
```

```
True
```

```
Clear[a, b, c, d]
mA = {{a, b}, {c, d}}; mA // MatrixForm
Inverse[mA] // MatrixForm
mA.Inverse[mA] // Simplify // MatrixForm
```

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

$$\begin{pmatrix} \frac{d}{-b\,c+a\,d} & -\frac{b}{-b\,c+a\,d} \\ -\frac{c}{-b\,c+a\,d} & \frac{a}{-b\,c+a\,d} \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

```
mA = {{1, 2}, {3, 4}};
"mA"
mA // MatrixForm
"transpose"
Transpose[mA] // MatrixForm
Inverse[mA] // MatrixForm
Inverse[mA].mA // MatrixForm
```

mA

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

transpose

$$\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$$

$$\begin{pmatrix} -2 & 1 \\ \frac{3}{2} & -\frac{1}{2} \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Creating Useful Matrices

Built In Matrix Creation Facilities

Some useful matrix-creation functions include IndentityMatrix, DiagonalMatrix, and ConstantArray.

```
mI = IdentityMatrix[2];
mD = DiagonalMatrix[{1, 2, 3}];
mC = ConstantArray[1, {2, 3}];
MatrixForm /@ {mI, mD, mC}
```

$$\left\{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}\right\}$$

Useful commands for matrix creation include RandomInteger and RandomReal. You can use the Dimensions command to retrieve the shape of your matrix. You can find the smallest and largest ele-

ments with Min and Max.

```
mRI = RandomInteger[100, {1000, 20}];
Dimensions[mRI]
{Min[mRI], Max[mRI]}
```

```
{1000, 20}
```

```
{0, 100}
```

The Table and Array commands are the basic list creation commands: Table takes an expression as input, and Array takes a function as input. Since a matrix is a just a list of lists, we can flexibly create matrices using Table. To create a matrix with the Table command, input an expression in two variables and iterators for those variables. For example, to create a 3 by 5 matrix of zeros we can do the following.

```
Table[0, {3}, {5}] // MatrixForm
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Here is the use of Array to accomplish the same thing:

```
Array[Function[{i, j}, 0], {3, 5}] // MatrixForm
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

More generally, we can index our iterators and produce any function of the indexes as matrix entries.

```
m3by5 = Table[(i - 1) * 5 + j, {i, 3}, {j, 5}] ;
m3by5 // MatrixForm
```

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{pmatrix}$$

Here we create the same matrix using Array.

```
m3by5 == Array[(#1 - 1) * 5 + #2 &, {3, 5}]
```

```
True
```

Indexing matrices is a little verbose in *Mathematica*: we must double our brackets to do indexing.

```
m3by5[[3]][[5]]
```

```
15
```

```
Array[a##1 &, {2, 2}]
Table[ai,j, {i, 2}, {j, 2}]
```

```
{{a1,1, a1,2}, {a2,1, a2,2}}
```

```
{{a1,1, a1,2}, {a2,1, a2,2}}
```

Stacking Matrices

Matrices can be stacked by making a “matrix of matrices” and then using the ArrayFlatten command. (If

the dimensions are unambiguous, you can use a constant to represent a constant matrix.)

```
mA = DiagonalMatrix[{1, 2}]; mB = DiagonalMatrix[{3, 4}];
ArrayFlatten[{{mA, mB}}] // MatrixForm
ArrayFlatten[{{mA}, {mB}}] // MatrixForm
ArrayFlatten[{{mA, a}, {0, mB}}] // MatrixForm
```

$$\begin{pmatrix} 1 & 0 & 3 & 0 \\ 0 & 2 & 0 & 4 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 \\ 0 & 2 \\ 3 & 0 \\ 0 & 4 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & a & a \\ 0 & 2 & a & a \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix}$$

Reshaping Matrices

```
mA = {Range[30]}; (* 1 by 30 matrix *)
mB = Partition[Flatten[mA], 10]; (* 3 by 10 matrix *)
mB // MatrixForm
Dimensions[mB]
```

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 & 30 \end{pmatrix}$$

```
{3, 10}
```

Creating Matrices

Some useful matrix-creation functions include IndentityMatrix, DiagonalMatrix, and ConstantArray.

```
mI = IdentityMatrix[2];
mD = DiagonalMatrix[{1, 2, 3}];
mC = ConstantArray[1, {2, 3}];
MatrixForm /@ {mI, mD, mC}
```

$$\left\{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}\right\}$$

Useful commands for matrix creation include RandomInteger and RandomReal. You can use the Dimensions command to retrieve the shape of your matrix. You can find the smallest and largest elements with Min and Max.

```
mRI = RandomInteger[100, {1000, 20}];
Dimensions[mRI]
{Min[mR], Max[mRI]}
```

```
{1000, 20}
```

```
{mR, 100}
```

The Table and Array commands are the basic list creation commands: Table takes an expression as input, and Array takes a function as input. Since a matrix is a just a list of lists, we can flexibly create matrices using Table. To create a matrix with the Table command, input an expression in two variables and iterators for those variables. For example, to create a 3 by 5 matrix of zeros we can do the following.

```
Table[0, {3}, {5}] // MatrixForm
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Here is the use of Array to accomplish the same thing:

```
Array[Function[{i, j}, 0], {3, 5}] // MatrixForm
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

More generally, we can index our iterators and produce any function of the indexes as matrix entries.

```
m3by5 = Table[(i - 1) * 5 + j, {i, 3}, {j, 5}] ;
m3by5 // MatrixForm
```

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{pmatrix}$$

Here we create the same matrix using Array.

```
m3by5 == Array[(#1 - 1) * 5 + #2 &, {3, 5}]
```

```
True
```

Indexing matrices is a little verbose in *Mathematica*: we must double our brackets to do indexing.

```
m3by5[[3]][[5]]
```

```
15
```

```
Array[a##&, {2, 2}]
Table[ai,j, {i, 2}, {j, 2}]
```

$$\{\{a_{1,1}, a_{1,2}\}, \{a_{2,1}, a_{2,2}\}\}$$

$$\{\{a_{1,1}, a_{1,2}\}, \{a_{2,1}, a_{2,2}\}\}$$

Reshaping Matrices

You can use the `Flatten` and `Partition` commands to reshape matrices.

```
mA = {Range[30]}; (* 1 by 30 matrix *)
mB = Partition[Flatten[mA], 10]; (* 3 by 10 matrix *)
mB // MatrixForm
Dimensions[mB]
```

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 & 30 \end{pmatrix}$$

```
{3, 10}
```

Stacking Matrices

Matrices can be stacked by making a “matrix of matrices” and then using the ArrayFlatten command.

```
mA = DiagonalMatrix[{1, 2}]; mB = DiagonalMatrix[{3, 4}];
ArrayFlatten[{{mA, mB}}] // MatrixForm
ArrayFlatten[{{mA}, {mB}}] // MatrixForm
```

$$\begin{pmatrix} 1 & 0 & 3 & 0 \\ 0 & 2 & 0 & 4 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 \\ 0 & 2 \\ 3 & 0 \\ 0 & 4 \end{pmatrix}$$

If the dimensions are unambiguous, you can conveniently use a constant to represent a constant matrix.

```
mD = ArrayFlatten[{{mA, a}, {0, mB}}]
% // MatrixForm
```

```
{{1, 0, a, a}, {0, 2, a, a}, {0, 0, 3, 0}, {0, 0, 0, 4}}
```

$$\begin{pmatrix} 1 & 0 & a & a \\ 0 & 2 & a & a \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix}$$

Unstacking Matrices

Stacks of square matrices can be unstacked using `Partition`.

```
mD = {{1, 0, a, a}, {0, 2, a, a}, {0, 0, 3, 0}, {0, 0, 0, 4}};
% // MatrixForm
mDu = Partition[mD, {2, 2}];
% // MatrixForm
mDu[[1, 1]] // MatrixForm
```

$$\begin{pmatrix} 1 & 0 & a & a \\ 0 & 2 & a & a \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix}$$

$$\begin{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix} & \begin{pmatrix} a & a \\ a & a \end{pmatrix} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 3 & 0 \\ 0 & 4 \end{pmatrix} \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$$

More complex unstacking can be handled by `PartitionRagged`.

```
? Internal`PartitionRagged
```

Info-4e55ad81-9cfa-46ee-9b60-b0f220c96db9

PartitionRagged[list, {n1,...,nk}] partitions list into ragged array with rows of length n1, ..., nk. PartitionRagged[array, {{n11,...,},...,{nm1,...}}] partitions depth m array along each dimension.

```
mDu2 = Internal`PartitionRagged[mD, {{3, 1}, {3, 1}}]
MatrixForm@Map[MatrixForm, mDu2, {2}]
mDu2[[1, 1]] // MatrixForm
```

```
{{{{1, 0, a}, {0, 2, a}, {0, 0, 3}}, {{a}, {a}, {0}}}, {{{0, 0, 0}}, {{4}}}}
```

$$\begin{pmatrix} \begin{pmatrix} 1 & 0 & a \\ 0 & 2 & a \\ 0 & 0 & 3 \end{pmatrix} & \begin{pmatrix} a \\ a \\ 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 4 \end{pmatrix} \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & a \\ 0 & 2 & a \\ 0 & 0 & 3 \end{pmatrix}$$

Linear Transformations in Two Dimensions

A 2 by 2 matrix can represent a tranformation or a point (x, y) in the Cartesian plane to another such point (x', y').

General Linear Transformation ($\mathbb{R}^2 \to \mathbb{R}^2$)

The following represents a general linear transformation of 2-vectors to 2-vectors.

noclass, nopdf

Dot::rect : Nonrectangular tensor encountered. >>

noclass, nopdf

$$\begin{pmatrix} 64 & 1 \\ 6 & \{0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09\} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} =$$

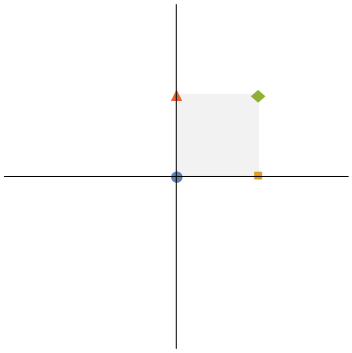
```
{{64, 1}, {6, {0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09}}}.{{x}, {y}}
```

Note that the zero vector always maps to the zero vector. We are going to visualize some special tranformations by looking at transformations of the points of the unit square. The unit square has corners at the following points: (0, 0), (1, 0), (1, 1), and (0, 1). Let us create a 2 by 4 matrix, where each column represents one of those corners, and then plot those four points.

noclass, nopdf

The resulting matrix is $\begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$, and we can plot the four points as follows.

noclass, nopdf



We can premultiply this representation of the unit square by any 2 by 2 matrix, and each column of the result represents a transformed point.

noclass, nopdf

$$\begin{pmatrix} 64 & 1 \\ 6 & \{0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09\} \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} =$$

```
{{64, 1}, {6, {0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09}}}.sqrpts
```

The first column of the result is of course the zero vector. The next is the first column of our transforation matrix. Then comes the sum of the two columns of our transformation matrix. And last, we find the second column of our transformation matrix.

Scale Transformations

Scale transformations uniformly scale all first coordinates and uniformly scale all second coordinates. Any diagonal matrix can be interpreted in terms of such scale transormations. The entries along the diagonal are called the scaling factors.

noclass, nopdf

```
{{sx, 0}, {0, sy}}
```

noclass, nopdf

Let $\begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix}$ be our two−dimensional scale−transformation matrix.

noclass, nopdf

$$\begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x\, s_x \\ y\, s_y \end{pmatrix}$$

noclass, nopdf

$$\begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & s_x & s_x & 0 \\ 0 & 0 & s_y & s_y \end{pmatrix}$$

Examples of Scale Transformations

nopdf, noclass

The matrix $\begin{pmatrix} 1.5 & 0 \\ 0 & 1 \end{pmatrix}$ will scale the horizontal values by 150%:

nopdf, noclass

nopdf, noclass

The matrix $\begin{pmatrix} 1 & 0 \\ 0 & 1.5 \end{pmatrix}$ will scale the vertical values by 150%:

nopdf, noclass

nopdf, noclass

The matrix $\begin{pmatrix} 1.5 & 0 \\ 0 & 1.5 \end{pmatrix}$ will scale the horizontal and vertical values by 150%:

nopdf, noclass

nopdf, noclass

The matrix $\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$ will scale the horizontal values by −100%:

nopdf, noclass

nopdf, noclass

The matrix $\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ will scale the vertical values by −100%:

nopdf, noclass

nopdf, noclass

The matrix $\begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$ will scale the horizontal and vertical values by −100%:

nopdf, noclass

Shear Transformations

A shear transformation in any direction proportionally displaces each point in that (signed) direction. Create a matrix for shearing as follows.

```
ClearAll[a, b, α, β]
mT = {{1, α}, {β, 1}};
MatrixForm[mT]
```

$$\begin{pmatrix} 1 & \alpha \\ \beta & 1 \end{pmatrix}$$

Here α determines the horizontal shearing, and β determines the vertical shearing.

noclass, nopdf

$$\begin{pmatrix} 1 & \alpha \\ \beta & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a + b\,\alpha \\ b + a\,\beta \end{pmatrix}$$

```
Map[MatrixForm, {mT, sqrpts}]
shear = mT.sqrpts; MatrixForm[shear]
{"horizontal shear transformation",
 mT /. {β → 0} // MatrixForm, (shear /. {β → 0}) // MatrixForm}
{"vertical shear transformation", mT /. {α → 0} // MatrixForm,
 shear /. {α → 0} // MatrixForm}
```

$$\left\{\begin{pmatrix} 1 & \alpha \\ \beta & 1 \end{pmatrix}, \text{sqrpts}\right\}$$

$$\{\{1, \alpha\}, \{\beta, 1\}\}.\text{sqrpts}$$

$$\left\{\text{horizontal shear transformation}, \begin{pmatrix} 1 & \alpha \\ 0 & 1 \end{pmatrix}, \{\{1, \alpha\}, \{0, 1\}\}.\text{sqrpts}\right\}$$

$$\left\{\text{vertical shear transformation}, \begin{pmatrix} 1 & 0 \\ \beta & 1 \end{pmatrix}, \{\{1, 0\}, \{\beta, 1\}\}.\text{sqrpts}\right\}$$

Examples of Shear Transformations

nopdf, noclass

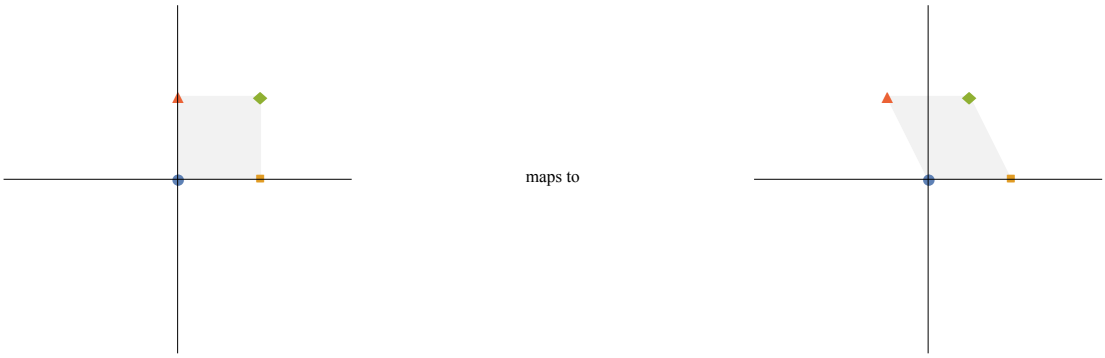
The matrix $\begin{pmatrix} 1 & 0.5 \\ 0 & 1 \end{pmatrix}$ produces a horizontal shear to the right:

nopdf, noclass

nopdf, noclass

The matrix $\begin{pmatrix} 1 & -0.5 \\ 0 & 1 \end{pmatrix}$ produces a horizontal shear to the left:

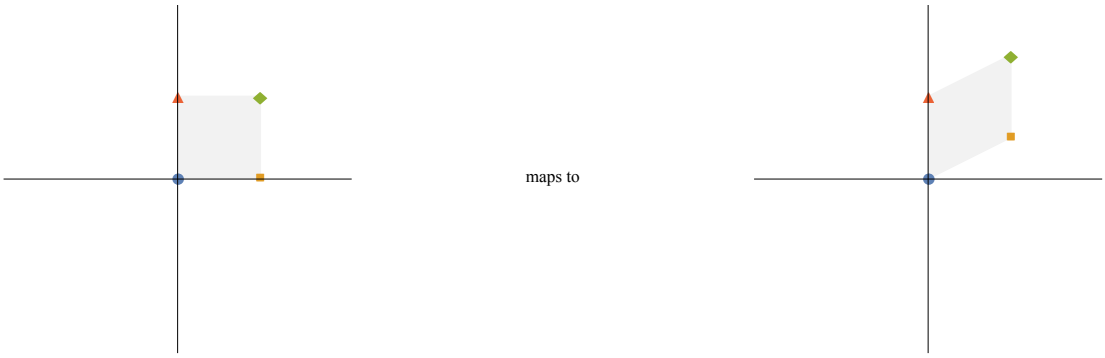
nopdf, noclass



nopdf, noclass

The matrix $\begin{pmatrix} 1 & 0 \\ 0.5 & 1 \end{pmatrix}$ produces a vertical shear upwards:

nopdf, noclass



nopdf, noclass

The matrix $\begin{pmatrix} 1 & 0 \\ -0.5 & 1 \end{pmatrix}$ produces a vertical shear downwards:

nopdf, noclass

Rotation Transformations

Suppose we want to map points in $\mathbb{R}^2$ to points rotated counter-clockwise by ninety degrees. We could do this by swapping the *x* and *y* coordinates and after changing the sign of the *y* coordinate. So the point (*a*, *b*) rotates to the point (–*b*, *a*). For example, (1, 2) would become (–2, 1). Note that the dot product of a point with its rotated point is 0. This is also true of any scalar multiple of the point with any scalar multiple of its rotation. So when two vectors are orthogonal, they have a dot product of 0.

```
Clear[θ]
{{Cos[θ], -Sin[θ]}, {Sin[θ], Cos[θ]}} // MatrixForm
```

$$\begin{pmatrix} \text{Cos}[\theta] & -\text{Sin}[\theta] \\ \text{Sin}[\theta] & \text{Cos}[\theta] \end{pmatrix}$$

```
rotate2d = Function[{pt2d, θ}, {{Cos[θ], -Sin[θ]}, {Sin[θ], Cos[θ]}}.pt2d]
pt1 = {0, 1};
pt2 = rotate2d[pt1, Pi / 2]
pt1.pt2
```

```
Function[{pt2d, θ}, {{Cos[θ], -Sin[θ]}, {Sin[θ], Cos[θ]}}.pt2d]
```

```
{-1, 0}
```

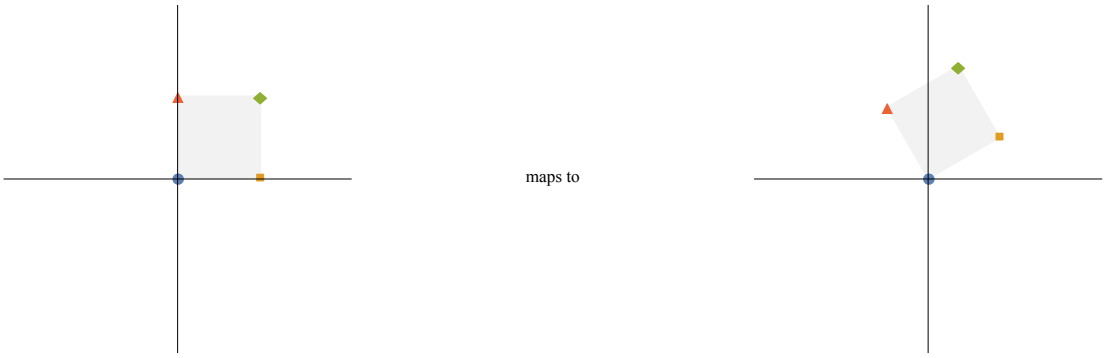
```
0
```

```
ClearAll[α, β]
mT = {{Cos[θ], -Sin[θ]}, {Sin[θ], Cos[θ]}};
mT // MatrixForm
"rotation transformation"
mT.sqrpts
% // MatrixForm
```

$$\begin{pmatrix} \text{Cos}[\theta] & -\text{Sin}[\theta] \\ \text{Sin}[\theta] & \text{Cos}[\theta] \end{pmatrix}$$

```
rotation transformation
{{Cos[θ], -Sin[θ]}, {Sin[θ], Cos[θ]}}.sqrpts
{{Cos[θ], -Sin[θ]}, {Sin[θ], Cos[θ]}}.sqrpts
```

The matrix $\begin{pmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{pmatrix}$ produces a 30 degree counterclockwise rotation:



The matrix $\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$ produces a 90 degree counterclockwise rotation:

The matrix $\begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$ produces a 180 degree counterclockwise rotation:

Miscellaneous

Trace and Determinant

Unusually for *Mathematica*, the trace and determinant operations have truncated names. (Moreover, *Mathematica*'s Trace command has nothing to do with matrices.)

```
Det[mA] (* matrix determinant *)
Tr[mA] (* matrix trace *)
```

```
-b c + a d
```

```
a + d
```

Linear Independence (redux)

A set of vectors *V* is said to be linearly independent if none of the vectors lies in the span of the others. Here are some ways to test for linear independence.

```
Clear[v1, v2, v3]
v1 = {1, 2, 3}; v2 = {4, 5, 6}; v3 = {7, 8, 9};
mM = Transpose[{v1, v2, v3}];
NullSpace[mM]
Det[mM]
```

```
{{1, -2, 1}}
```

```
0
```

Suppose *V* is a finite set of N-vectors. We want to determine whether a vector *b* is in the span of *V*. Create a matrix *M* with columns that are the vectors in *V*, and ask whether the equation *M*.*x* = *b* has a solution.

```
Clear[v1, v2, v3, mM, b]
v1 = {1, 2, 3}; v2 = {4, 5, 6}; v3 = {7, 8, 9}; b = {10, 11, 12};
mM = Transpose[{v1, v2, v3}];
LinearSolve[mM, b]
```

```
{-2, 3, 0}
```

Note that LinearSolve only returns a single solution, not a description of all the possible solutions. For that, you need to use Solve:

```
Clear[v1, v2, v3, mM, b]
v1 = {1, 2, 3}; v2 = {4, 5, 6}; v3 = {7, 8, 9}; b = {10, 11, 12};
mM = Transpose[{v1, v2, v3}];
Solve[mM.{x1, x2, x3} == b, {x1, x2, x3}]
```

```
Solve::ivar : 2 is not a valid variable. >>
```

```
Solve[{6 + 7 x3, 9 + 8 x3, 12 + 9 x3} == {10, 11, 12}, {2, 1, x3}]
```

```
NullSpace[mM]
```

```
{{1, -2, 1}}
```

```
mM.%
```

Dot::dotsh : Tensors {{1, 4, 7}, {2, 5, 8}, {3, 6, 9}} and {{1, –2, 1}} have incompatible shapes. >>

```
{{1, 4, 7}, {2, 5, 8}, {3, 6, 9}}.{{1, -2, 1}}
```

Matrix-Oriented Methods of Testing for Linear Dependence

```
Clear[v1, v2, v3]
v1 = {1, 2, 3}; v2 = {4, 5, 6}; v3 = {7, 8, 9};
mM = Transpose[{v1, v2, v3}];
NullSpace[mM]
Det[mM]
```

```
{{1, -2, 1}}
```

```
0
```

Suppose *V* is a finite set of N-vectors. We want to determine whether a vector *b* is in the span of *V*. Create a matrix *M* with columns that are the vectors in *V*, and ask whether the equation *M*.*x* = *b* has a solution.

```
Clear[v1, v2, v3, mM, b]
v1 = {1, 2, 3}; v2 = {4, 5, 6}; v3 = {7, 8, 9}; b = {10, 11, 12};
mM = Transpose[{v1, v2, v3}];
LinearSolve[mM, b]
```

```
{-2, 3, 0}
```

Note that LinearSolve only returns a single solution, not a description of all the possible solutions. For that, you need to use Solve:

```
Clear[v1, v2, v3, mM, b]
v1 = {1, 2, 3}; v2 = {4, 5, 6}; v3 = {7, 8, 9}; b = {10, 11, 12};
mM = Transpose[{v1, v2, v3}];
Solve[mM.{x1, x2, x3} == b, {x1, x2, x3}]
```

Solve::ivar : 2 is not a valid variable. >>

```
Solve[{6 + 7 x3, 9 + 8 x3, 12 + 9 x3} == {10, 11, 12}, {2, 1, x3}]
```

```
NullSpace[mM]
```

```
{{1, -2, 1}}
```

```
mM.%
```

Dot::dotsh : Tensors {{1, 4, 7}, {2, 5, 8}, {3, 6, 9}} and {{1, –2, 1}} have incompatible shapes. >>

```
{{1, 4, 7}, {2, 5, 8}, {3, 6, 9}}.{{1, -2, 1}}
```

Prefix and Postfix Notation

Most of the time we will use the familiar prefix notation for *Mathematica* functions, as above. But at times it can be convenient to use an alternative prefix or postfix notation.

```
mA = {{1, 2}, {3, 4}};
mA // Transpose
mA // Inverse
Transpose@mA
Inverse@mA
```

```
{{1, 3}, {2, 4}}
```

$$\left\{\{-2, 1\}, \left\{\frac{3}{2}, -\frac{1}{2}\right\}\right\}$$

```
{{1, 3}, {2, 4}}
```

$$\left\{\{-2, 1\}, \left\{\frac{3}{2}, -\frac{1}{2}\right\}\right\}$$

The main convenience from the postfix notation comes in readability when a sequence of operations is chained.

```
mA = {{1, 2}, {3, 4}};
(mA // Inverse // Transpose) === (mA // Transpose // Inverse)
```

```
True
```

Eigenvalues and Eigenvectors

Given a matrix A, and eigenvalue is a scalar λ such that the matrix $(A - \lambda \mathrm{I})$ is singular. We can find such a scalar by solving the characteristic equation $|A - \lambda \mathrm{I}| = 0$. For example, consider the matrix

$$\mathrm{A} = \begin{pmatrix} 4 & -1 \\ -3 & 2 \end{pmatrix} \Rightarrow \mathrm{A} - \lambda \mathrm{I} = \begin{pmatrix} 4-\lambda & -1 \\ -3 & 2-\lambda \end{pmatrix}$$

The characteristic polynomial is $|A - \lambda \mathrm{I}|$, which is $\lambda^2 - 6\lambda + 5$. This is just an ordinary polynomial in λ. the characteristic equation $\lambda^2 - 6\lambda + 5 = 0$. Applying the quadratic equation, we get solutions $\lambda_1, \lambda_2 = 1, 5$. We can use *Mathematica* to implement exactly these steps.

```
Clear[λ]
mA = {{4, -1}, {-3, 2}}
mL = mA - λ * IdentityMatrix[2]
charpoly = Det[mL]
soln = Solve[charpoly == 0, λ]
```

```
{{4, -1}, {-3, 2}}
```

$$\{\{4-\lambda, -1\}, \{-3, 2-\lambda\}\}$$

$$5 - 6\lambda + \lambda^2$$

$$\{\{\lambda \to 1\}, \{\lambda \to 5\}\}$$

Mathematica also offers some special commands that allow us to proceed more concisely. Read the documentation for `CharacteristicPolynomial` and `Eigenvalues`.

```
CharacteristicPolynomial[mA, λ]
Eigenvalues[mA]
```

$5 - 6\,\lambda + \lambda^2$

{5, 1}

With each eigenvalue λ of a matrix A, we can associate an eigenvector v, such that

$Av = \lambda v$

In other words, premultiplying A times one of its eigenvectors produces the same outcome as scaling that vector by the associated eigenvalue. Clearly if v is an eigenvector, so is any nonzero scalar multiple of v. Eigenvectors are not unique.

```
Clear[x1, x2]
mL1 = mL /. soln[[1]]; mL2 = mL /. soln[[2]];
Solve[mL1.{{x1}, {x2}} == 0, {x1, x2}]
```

Solve::svars : Equations may not give solutions for all "solve" variables. ≫

{{x2 → 3 x1}}

```
Solve[mL1.{{x1}, {x2}} == 0, {x1, x2}] /. {x1 → 1}
Solve[mL2.{{x1}, {x2}} == 0, {x1, x2}] /. {x1 → 1}
```

Solve::svars : Equations may not give solutions for all "solve" variables. ≫

{{x2 → 3}}

Solve::svars : Equations may not give solutions for all "solve" variables. ≫

{{x2 → -1}}

Once again, *Mathematica* offers some specialized commands for exploring eigensystems. The `Eigenvectors` command produces eigenvectors (naturally enough), while the `Eigensystem` command returns both the eigenvalues and associated eigenvectors.

```
Clear[λ]
mB = 2 * {{1, 0, 2}, {0, 5, 0}, {3, 0, 2}};
mB // MatrixForm
cpB = CharacteristicPolynomial[mB, λ]
m1 = mB + 2 * IdentityMatrix[3]
m1 // MatrixForm
Solve[m1.{x1, x2, x3} == 0, {x1, x2, x3}]
Factor[cpB]
Eigensystem[mB]
evec = {{1}, {0}, {-1}}
mB.evec == -1 * evec
```

$$\begin{pmatrix} 2 & 0 & 4 \\ 0 & 10 & 0 \\ 6 & 0 & 4 \end{pmatrix}$$

$$-160 - 44\,\lambda + 16\,\lambda^2 - \lambda^3$$

```
{{4, 0, 4}, {0, 12, 0}, {6, 0, 6}}
```

$$\begin{pmatrix} 4 & 0 & 4 \\ 0 & 12 & 0 \\ 6 & 0 & 6 \end{pmatrix}$$

Solve::svars : Equations may not give solutions for all "solve" variables. >>

```
{{x2 → 0, x3 → -x1}}
```

$$-(-10+\lambda)\,(-8+\lambda)\,(2+\lambda)$$

```
{{10, 8, -2}, {{0, 1, 0}, {2, 0, 3}, {-1, 0, 1}}}
{{1}, {0}, {-1}}
False
Eigenvectors[mA]
Eigensystem[mA]
{{-1, 1}, {1, 3}}
{{5, 1}, {{-1, 1}, {1, 3}}}
mP = {{1, 1}, {3, -1}}
Inverse[mP].mA.mP
{{1, 1}, {3, -1}}
{{1, 0}, {0, 5}}
```

Linear Comparative Statics

Equations

Here is a very simple “IS-LM” description of goods and money market equilibrium in the macroeconomy.

$$\begin{aligned} y &= g - \beta\, r \\ m - p &= \mu_1\, y - \mu_2\, (r + \pi) \end{aligned} \tag{1}$$

Here r is the real interest rate, y is (the log of) real income, $m - p$ is the log of the real money supply, π is the expected rate of inflation, and g is a fiscal stance variable.

In *Mathematica*, we need to distinguish different uses of equals signs: use a single equals sign for assignment (Set) and two to create an equation (Equal). We can create an equation and assign it to a name.

```
Clear[y, g, r, m, p, Π]
iseq = y == g - β * r;
lmeq = m - p == μ1 * y - μ2 * (r + Π);
```

Later, we can use these names to refer to our equations.

```
iseq
lmeq
```

$$y == g - r\,\beta$$

$$m - p == y\,\mu 1 - \mu 2\,(r + \Pi)$$

Note that your symbols may be re-ordered when output.

“Keynesian” Model

We turn these equations into a “Keynesian” model by designating r and y to be the endogenous variables. Once we have a specification of the endogenous variables for a system of equations, we can attempt to solve it. A “solution” of the model will express r and y as functions of the exogenous variables (m, g, and π) and the structural parameters. Such a solution is also called a “reduced form”.

Use of the Solve Command

```
soln01K = Solve[iseq && lmeq, {y, r}]
```

$$\left\{\left\{y \to -\frac{-m\,\beta + p\,\beta - g\,\mu 2 - \beta\,\mu 2\,\Pi}{\beta\,\mu 1 + \mu 2},\ r \to -\frac{m - p - g\,\mu 1 + \mu 2\,\Pi}{\beta\,\mu 1 + \mu 2}\right\}\right\}$$

Mathematica represents a solution as a collection of rules. Note that it is a list of lists of solutions, since some systems handled by Solve may have multiple solutions. Get the first solution by indexing this list. (*Mathematica* indexes with double brackets and uses unit-based indexing.)

```
soln01K = soln01K[[1]]
```

$$\left\{y \to -\frac{-m\,\beta + p\,\beta - g\,\mu 2 - \beta\,\mu 2\,\Pi}{\beta\,\mu 1 + \mu 2},\ r \to -\frac{m - p - g\,\mu 1 + \mu 2\,\Pi}{\beta\,\mu 1 + \mu 2}\right\}$$

A Matrix Algebra Approach

Next we use a more explicit solution procedure, which can be very helpful in understanding what is going on. Gather the endogenous variables on the left hand side of the equations. Also, we make explicit the implicit coefficient of unity on y.

First recall our IS and LM equations.

```
iseq
lmeq
```

```
y == g - r β
```

```
m - p == y μ1 - μ2 (r + Π)
```

$$\begin{aligned} 1\, y + \beta\, r &= g \\ \mu 1\, y - \mu 2\, r &= m - p + \mu 2\, \pi \end{aligned} \tag{2}$$

We are going to introduce a useful shorthand for writing this system of equations. Write these two equations as

$$\begin{bmatrix} 1 & \beta \end{bmatrix} \begin{bmatrix} y \\ r \end{bmatrix} = [g]$$

$$\begin{bmatrix} \mu_1 & -\mu_2 \end{bmatrix} \begin{bmatrix} y \\ r \end{bmatrix} = [m - p + \mu_2\, \pi] \tag{3}$$

Now "stack" these two equations to set up a matrix equation in the form $J\, x = b$:

$$\begin{bmatrix} 1 & \beta \\ \mu_1 & -\mu_2 \end{bmatrix} \begin{bmatrix} y \\ r \end{bmatrix} = \begin{bmatrix} g \\ m - p + \mu_2\, \pi \end{bmatrix} \tag{4}$$

The 2×2 coefficient matrix that premultiplies the endogenous variables is known as the Jacobian matrix. Note how each row of the Jacobian matrix contains parameters from a single equation.

```
mJ = {{1, β}, {μ1, -μ2}}; (* Jacobian matrix *)
mJ // MatrixForm (* Display mJ in matrix form *)
```

$$\begin{pmatrix} 1 & \beta \\ \mu 1 & -\mu 2 \end{pmatrix}$$

Suppose that the Jacobian matrix has a "multiplicative inverse". In this case it does:

```
Inverse[mJ] // Simplify // MatrixForm
```

$$\begin{pmatrix} \frac{\mu 2}{\beta\, \mu 1 + \mu 2} & \frac{\beta}{\beta\, \mu 1 + \mu 2} \\ \frac{\mu 1}{\beta\, \mu 1 + \mu 2} & -\frac{1}{\beta\, \mu 1 + \mu 2} \end{pmatrix}$$

If we multiply both sides of this equation by the inverse of the Jacobian matrix, we produce the solution or "reduced form" of the model. Recall our exogenous vector is

```
b = {{g}, {m - p + μ2 * Π}};
b // MatrixForm
```

$$\begin{pmatrix} g \\ m - p + \mu 2\ \Pi \end{pmatrix}$$

So produce your solution for *y* and *r* by premultiplying the exogenous vector by the inverse of the coefficient matrix:

```
soln02K = Inverse[mJ].b;
soln02K // FullSimplify // MatrixForm
```

$$\begin{pmatrix} \frac{g\ \mu 2+\beta\ (m-p+\mu 2\ \Pi)}{\beta\ \mu 1+\mu 2} \\ \frac{-m+p+g\ \mu 1-\mu 2\ \Pi}{\beta\ \mu 1+\mu 2} \end{pmatrix}$$

In more detail:

$$\begin{bmatrix} y \\ r \end{bmatrix} = \frac{-1}{\mu_2 + \beta\ \mu_1} \begin{bmatrix} -\mu_2 & -\beta \\ -\mu_1 & 1 \end{bmatrix} \begin{bmatrix} g \\ m + \mu_2\ \pi \end{bmatrix} = \begin{pmatrix} \frac{g\ \mu_2+\beta(m-p+\mu_2\ \Pi)}{\mu 2+\beta\ \mu 1} \\ -\frac{m-p-g\ \mu_1+\mu_2\ \Pi}{\mu_2+\beta\ \mu_1} \end{pmatrix} \tag{5}$$

This is nice and explicit. Computationally, however, we often do not want to use an explicit inverse. Instead we can use `LinearSolve` or `Solve`.

Using LinearSolve (or Solve)

```
soln03K = LinearSolve[mJ, b] // MatrixForm
```

$$\begin{pmatrix} \frac{m\ \beta-p\ \beta+g\ \mu 2+\beta\ \mu 2\ \Pi}{\beta\ \mu 1+\mu 2} \\ \frac{-m+p+g\ \mu 1-\mu 2\ \Pi}{\beta\ \mu 1+\mu 2} \end{pmatrix}$$

```
soln04K = Solve[mJ.{{y}, {r}} == b, {y, r}][[1]]
```

$$\left\{y \to -\frac{-m\ \beta + p\ \beta - g\ \mu 2 - \beta\ \mu 2\ \Pi}{\beta\ \mu 1 + \mu 2},\ r \to -\frac{m - p - g\ \mu 1 + \mu 2\ \Pi}{\beta\ \mu 1 + \mu 2}\right\}$$

Note that we run into problems if we try to treat a column vector as a variable.

```
Clear[x]
x = {{y}, {r}};
Solve[mJ.x == b, x]
```

Solve::ivar : {y} is not a valid variable. >>

```
Solve[{{y + r β}, {y μ1 - r μ2}} == {{g}, {m - p + μ2 Π}}, {{y}, {r}}]
```

However, because Dot supports dimension reduction, we can do something very similar.

Using Dot across Differing Dimensions

```
mJ = {{1, β}, {μ1, -μ2}}; (* two dimensional *)
x = {y, r}; (* one dimensional *)
b = {g, m - p + μ2 * Π}; (* one dimensional *)
soln05K = Solve[mJ.x == b, x][[1]]
```

$$\left\{y \to -\frac{-m\,\beta + p\,\beta - g\,\mu 2 - \beta\,\mu 2\,\Pi}{\beta\,\mu 1 + \mu 2},\ r \to -\frac{m - p - g\,\mu 1 + \mu 2\,\Pi}{\beta\,\mu 1 + \mu 2}\right\}$$

Comparative Statics

Once we have our reduced form, we are ready to look at the comparative statics of the model. We will use the partial derivative command (D) and the ReplaceAll command (/.).

Let us begin by looking at the response of the solutions to a change in *g*.

```
D[y /. soln05K, g]
D[r /. soln05K, g]
D[{y, r} /. soln05K, g]
```

$$\frac{\mu 2}{\beta\,\mu 1 + \mu 2}$$

$$\frac{\mu 1}{\beta\,\mu 1 + \mu 2}$$

$$\left\{\frac{\mu 2}{\beta\,\mu 1 + \mu 2}, \frac{\mu 1}{\beta\,\mu 1 + \mu 2}\right\}$$

```
cs01K = D[{y, r} /. soln05K, {{m, p, Π, g}}]
```

$$\left\{\left\{\frac{\beta}{\beta\,\mu 1 + \mu 2}, -\frac{\beta}{\beta\,\mu 1 + \mu 2}, \frac{\beta\,\mu 2}{\beta\,\mu 1 + \mu 2}, \frac{\mu 2}{\beta\,\mu 1 + \mu 2}\right\}, \left\{-\frac{1}{\beta\,\mu 1 + \mu 2}, \frac{1}{\beta\,\mu 1 + \mu 2}, -\frac{\mu 2}{\beta\,\mu 1 + \mu 2}, \frac{\mu 1}{\beta\,\mu 1 + \mu 2}\right\}\right\}$$

This can be easier to look at in a table.

```
TableForm[cs01K, TableHeadings → {{"y", "r"}, {"m", "p", "Π", "g"}}]
```

	m	p	Π	g
y	$\frac{\beta}{\beta\,\mu 1+\mu 2}$	$-\frac{\beta}{\beta\,\mu 1+\mu 2}$	$\frac{\beta\,\mu 2}{\beta\,\mu 1+\mu 2}$	$\frac{\mu 2}{\beta\,\mu 1+\mu 2}$
r	$-\frac{1}{\beta\,\mu 1+\mu 2}$	$\frac{1}{\beta\,\mu 1+\mu 2}$	$-\frac{\mu 2}{\beta\,\mu 1+\mu 2}$	$\frac{\mu 1}{\beta\,\mu 1+\mu 2}$

Signing the Comparative Statics

```
cs01Ksigns = Assuming[β > 0 && μ1 > 0 && μ2 > 0,
   Simplify[Sign[cs01K]]
 ]
TableForm[cs01Ksigns /. {-1 → "-", 1 → "+"},
 TableHeadings → {{"y", "r"}, {"m", "p", "Π", "g"}}]
```

```
{{1, -1, 1, 1}, {-1, 1, -1, 1}}
```

	m	p	Π	g
y	+	−	+	+
r	−	+	−	+

"Classical" Model

Here is another example of comparative statics experiments in a simple linear model. We work with the same structural equations, but we produce a stylized "Classical" model by specifying that m and i are endogenous.

Solution using Solve

Let us recall our "structural" equations:

```
iseq
lmeq
```

```
y == g - r β
```

```
m - p == y μ1 - μ2 (r + Π)
```

Naturally we can simply solve the same equations for the new endogenous variables.

```
soln01C = Solve[iseq && lmeq, {p, r}][[1]]
```

$$\left\{p \to -\frac{-m\,\beta + y\,\beta\,\mu 1 - g\,\mu 2 + y\,\mu 2 - \beta\,\mu 2\,\Pi}{\beta},\ r \to -\frac{-g + y}{\beta}\right\}$$

More Explicit Solution (using Matrix Algebra)

But let us again be a bit more explicit. In preparation, let us gather the endogenous variables on the left hand side of the equations.

$$\begin{aligned} \beta\, r &= g - y \\ \mu 2\, r - p &= -m + \mu 1\, y - \mu 2\, \pi \end{aligned} \tag{6}$$

Note the recursive structure of the model: to solve for the interest rate, we only need the first equation. We can then plug this solution for i into the second equation to solve for m.

Let us make this yet more explicit by including coefficient of 0 or 1 where appropriate:

$$\begin{aligned} \beta\, r + 0 \cdot p &= \text{g} - \text{y} \\ \mu_2\, r - 1 \cdot p &= -\text{m} + \mu_1\, y - \mu_2\, \pi \end{aligned} \tag{7}$$

Now set up the matrix equation in the form $J\, x = b$:

$$\begin{bmatrix} \beta & 0 \\ \mu_2 & -1 \end{bmatrix} \begin{bmatrix} r \\ p \end{bmatrix} = \begin{bmatrix} g - y \\ -m + \mu_1\, y - \mu_2\, \pi \end{bmatrix} \tag{8}$$

Now the coefficient matrix is

```
cJ = {{β, 0}, {μ2, -1}}; cJ // MatrixForm
```

$$\begin{pmatrix} \beta & 0 \\ \mu 2 & -1 \end{pmatrix}$$

The inverse is

```
Inverse[cJ] // MatrixForm
```

$$\begin{pmatrix} \frac{1}{\beta} & 0 \\ \frac{\mu 2}{\beta} & -1 \end{pmatrix}$$

Once again, we can multiply both sides of this equation by the inverse of the Jacobian matrix to produce the reduced form of the model.

$$\begin{bmatrix} r \\ p \end{bmatrix} = \frac{1}{\beta} \begin{bmatrix} 1 & 0 \\ \mu_2 & -\beta \end{bmatrix} \begin{bmatrix} g - y \\ -\text{m} + \mu_1\, y - \mu_2\, \pi \end{bmatrix} \tag{9}$$

```
Clear[g]
Inverse[cJ].{{g - y}, {-m + μ1 y - μ2 Π}} // MatrixForm
```

$$\begin{pmatrix} \frac{\text{g} - \text{y}}{\beta} \\ \text{m} - \text{y}\, \mu 1 + \frac{(\text{g} - \text{y})\, \mu 2}{\beta} + \mu 2\, \Pi \end{pmatrix}$$

Solution using LinearSolve

```
solnC02 = LinearSolve[cJ, {g - y, -m + μ1 * y - μ2 * Π}] // Expand
```

$$\left\{ \frac{\text{g}}{\beta} - \frac{\text{y}}{\beta},\ \text{m} - \text{y}\, \mu 1 + \frac{\text{g}\, \mu 2}{\beta} - \frac{\text{y}\, \mu 2}{\beta} + \mu 2\, \Pi \right\}$$

Comparative Statics

```
csC01 = D[solnC02, {{m, Π, y, g}}]
TableForm[csC01, TableHeadings → {{"r", "p"}, {"m", "Π", "y", "g"}}]
```

$$\left\{ \left\{ 0,\ 0,\ -\frac{1}{\beta},\ \frac{1}{\beta} \right\},\ \left\{ 1,\ \mu 2,\ -\mu 1 - \frac{\mu 2}{\beta},\ \frac{\mu 2}{\beta} \right\} \right\}$$

	m	Π	y	g
r	0	0	$-\frac{1}{\beta}$	$\frac{1}{\beta}$
p	1	$\mu 2$	$-\mu 1 - \frac{\mu 2}{\beta}$	$\frac{\mu 2}{\beta}$

Comparative Statics: Signs

```
csC01signs = Assuming[β > 0 && μ1 > 0 && μ2 > 0,
   Simplify[Sign[csC01]]
 ]
TableForm[csC01signs /. {-1 → "-", 1 → "+"},
 TableHeadings → {{"r", "p"}, {"m", "Π", "y", "g"}}]
```

```
{{0, 0, -1, 1}, {1, 1, -1, 1}}
```

	m	Π	y	g
r	0	0	-	+
p	+	+	-	+

"Post Keynesian" Model (Exercise)

Play the "one structure for multiple models" game one more time. Work with the same structural equations. This time create a stylized "Post Keynesian" model by specifying that y and m are the endogenous variables.

Differential Calculus and Nonlinear Comparative Statics

Univariate Differential Calculus

Difference Quotients

```
Clear["Global`*"] (* clear all global symbols *)
f = Function[x, x^2];
dq = (f[x + Δx] - f[x]) / Δx
Simplify[dq]
```

$$\frac{-x^2 + (x + \Delta x)^2}{\Delta x}$$

$$2\,x + \Delta x$$

```
Limit[dq, Δx → 0]
```

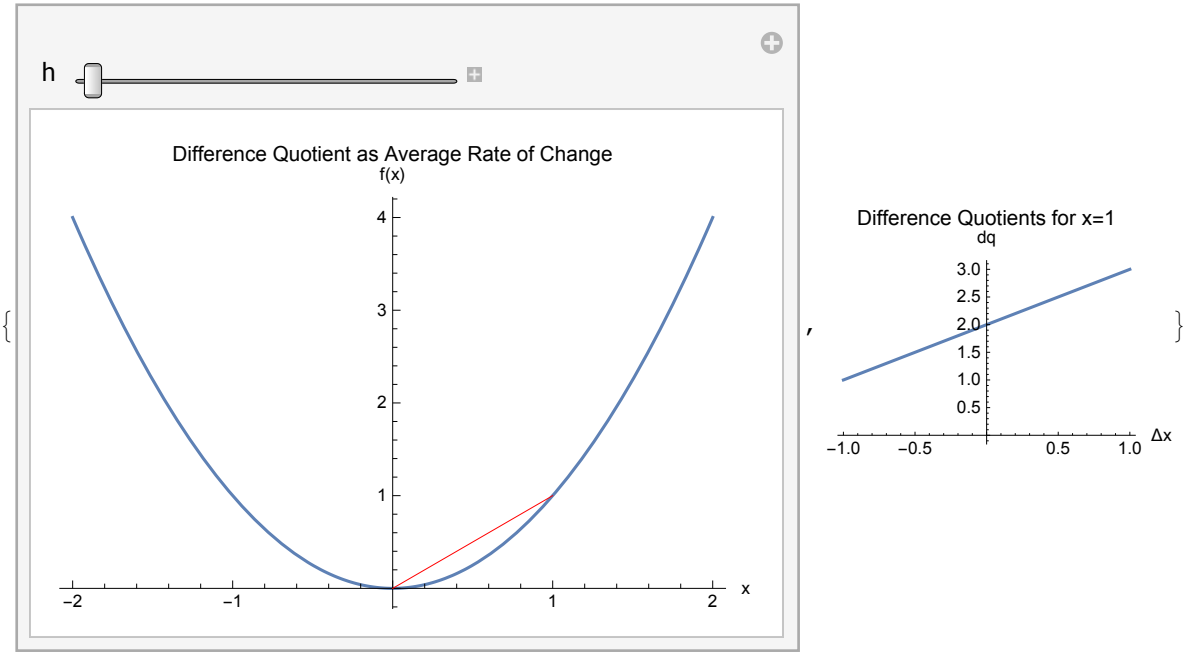
$$2\,x$$

```
D[f[x], x]
```

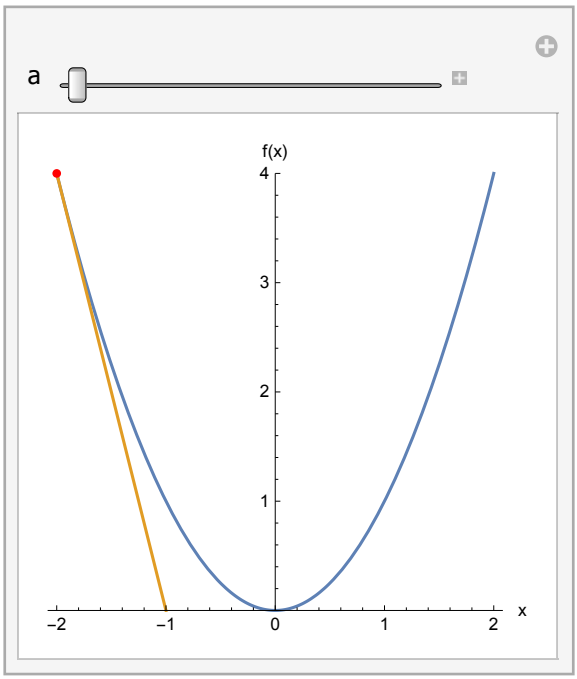
$$2\,x$$

Average Rate of Change

```
ClearAll[f, x, x0, Δx]
f[x_] := x^2; dfdx = (f[x + Δx] - f[x]) / Δx; x0 = 1;
{Manipulate[Plot[f[x], {x, -2, 2}, AxesLabel → {"x", "f(x)"},
    PlotLabel → "Difference Quotient as Average Rate of Change",
    Epilog → {Directive[Red], Line[{{x0, f[x0]}, {x0 + h, f[x0 + h]}}]}], {h, -1, 1}],
 Plot[dfdx /. {x → 1, Δx → h}, {h, -1, 1}, AxesOrigin → {0, 0},
  AxesLabel → {"Δx", "dq"}, PlotLabel → "Difference Quotients for x=1"]}
```



```
Manipulate[Plot[{x^2, a * a + 2 * a (x - a)}, {x, -2, 2}, PlotRange → {0, 4},
  AxesLabel → {"x", "f(x)"}, AspectRatio → Automatic, ImageSize → 250,
  Epilog → {PointSize[Medium], Red, Point[{a, a * a}]}], {a, -2, 2}]
```



Derivative

When it exists, we define the derivative as the limiting value of the difference quotient.

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

```
Clear[dq, f, x, Δx]
f = Function[x, x^2];
dq = (f[x + Δx] - f[x]) / Δx
Limit[dq, Δx → 0]
```

$$\frac{-x^2 + (x + \Delta x)^2}{\Delta x}$$

$$2x$$

Mathematica allows very natural notation for the derivative.

```
ClearAll[f, x]
f = x ↦ x^2;
f'[x]
```

$$2x$$

```
g = x ↦ x^3;
g'[x]
```

$$3\,x^2$$

Rules of Differentiation

```
(* illustrate sum rule *)
Clear[f, g, h, x]
h[x_] = f[x] + g[x];
h'[x]
```

$$f'[x] + g'[x]$$

```
(* illustrate product rule *)
Clear[f, g, h, x]
h[x_] = f[x] * g[x];
h'[x]
```

$$g[x]\,f'[x] + f[x]\,g'[x]$$

Taylor Series

Mathematica can produce a Taylor series expansion of an arbitrary function *f* around a point *p*.

```
Series[f[x], {x, p, 3}]
```

$$f[p] + f'[p]\,(x-p) + \frac{1}{2}\,f''[p]\,(x-p)^2 + \frac{1}{6}\,f^{(3)}[p]\,(x-p)^3 + O[x-p]^4$$

You can also do this for specified functions.

```
expSeries = Series[Exp[x], {x, 0, 3}]
```

$$1 + x + \frac{x^2}{2} + \frac{x^3}{6} + O[x]^4$$

```
CoefficientList[expSeries, x]
```

$$\left\{1,\ 1,\ \frac{1}{2},\ \frac{1}{6}\right\}$$

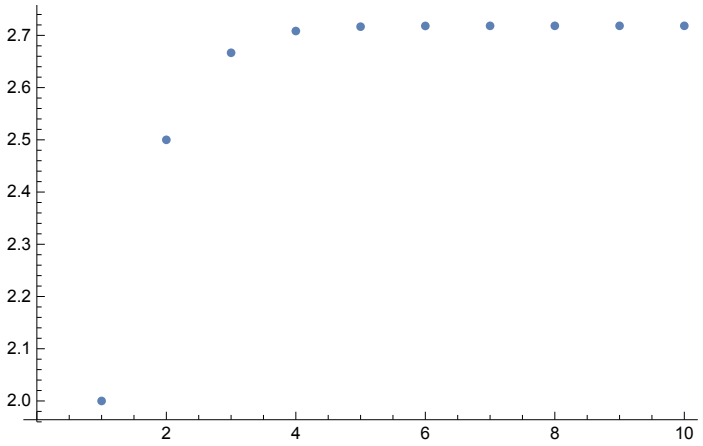
The output of the Series command is a SeriesData object. (Use the the InputForm or FullForm command to examine it.) If you need a normal expression, use the Normal command.

```
Normal[expSeries]
```

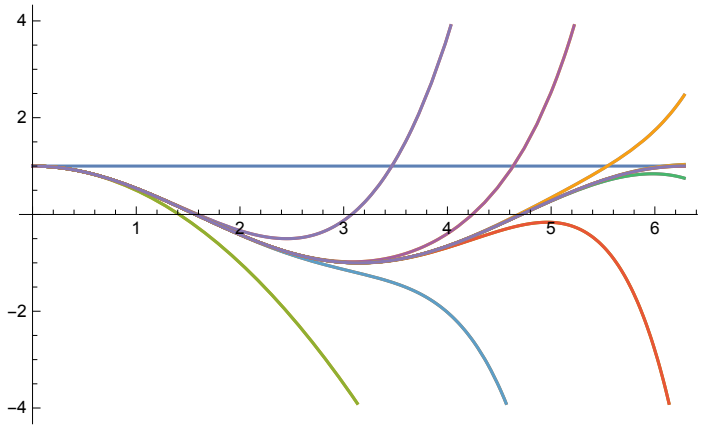
$$1 + x + \frac{x^2}{2} + \frac{x^3}{6}$$

```
Map[
    Function[{k}, Normal[Series[Exp[x], {x, 0, k}]]],
    Range[10]
  ] /. {x → 1} // N
ListPlot[%]
```

```
{2., 2.5, 2.66667, 2.70833, 2.71667, 2.71806, 2.71825, 2.71828, 2.71828, 2.71828}
```




```
(* Interesting example from Mathematica documentation *)
(* Note use of Normal; cannot directly plot a Series *)
Plot[Evaluate[Table[Normal[Series[Cos[x], {x, 0, n}]], {n, 20}]], {x, 0, 2 Pi}]
```



Multivariate Differential Calculus

The gradient of a function is perpendicular to any curve in the level set. In this section we will illustrate this. First we illustrate the gradient.

```
Simplify[Log[cd[n, k]] == Log[10] + 0.3 Log[k] + 0.7 Log[n], n > 0 && k > 0]
```

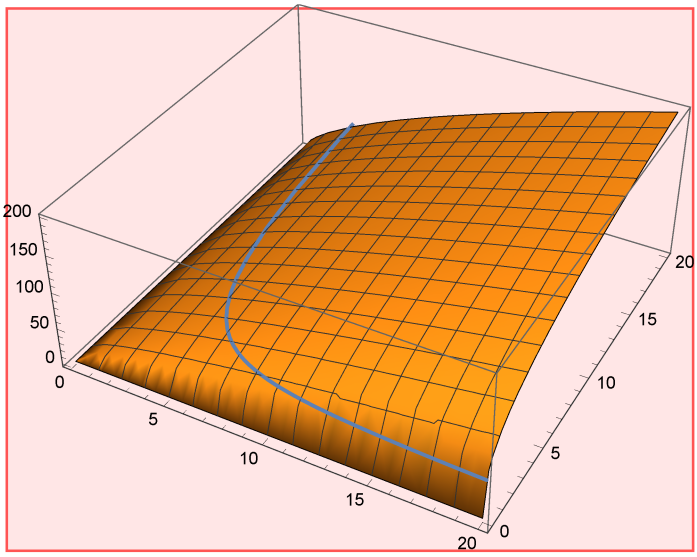
```
Log[cd[n, k]] == Log[10] + 0.3 Log[k] + 0.7 Log[n]
```

```
ClearAll[y, n, k, cd]
cd = Function[{n, k}, 10 * n^0.7 * k^0.3]
{dydn, dydk} = D[cd[n, k], {{n, k}}]
nkstart = {n → 5, k → 5}
"output at (5,5):"
cdstart = cd[n, k] /. nkstart
"the gradient at (5,5):"
{dydn, dydk} /. nkstart
pt1 = {n, k, cd[n, k]} /. nkstart
pt2 = {n + dn, k + dk, cd[n + dydn, k + dydk]} /. nkstart
g1 = Plot3D[cd[n, k], {n, 0, 20}, {k, 0, 20}];
g2 = Graphics3D[{Arrowheads[0.025, Appearance → "Projected"],
     Arrow[Tube[{pt1, pt2}]]}, PlotRange → {{0, 20}, {0, 20}, {0, 200}}];
kyf = Function[{n, yf}, (yf / 10 / n^0.7) ^ (1 / 0.3)];
g3 = ParametricPlot3D[{n, kyf[n, cdstart], cdstart},
    {n, 0, 20}, PlotRange → {{0, 20}, {0, 20}, {0, 200}}];
Show[{g1, g2, g3}]
"2d projection"
g1 = ContourPlot[cd[n, k] == cdstart, {n, 0, 20}, {k, 0, 20},
  PlotPoints → 100, Epilog → Arrow[{{n, k}, {n + dydn, k + dydk}} /. nkstart]]
tangentSlope = D[kyf[n, cdstart], n] /. nkstart
g2 = ParametricPlot[{n, kyf[5, cdstart] + tangentSlope * (n - 5)},
  {n, 0.1, 7}, PlotStyle → {Thin, Dashed}]
Show[{g1, g2}]
```

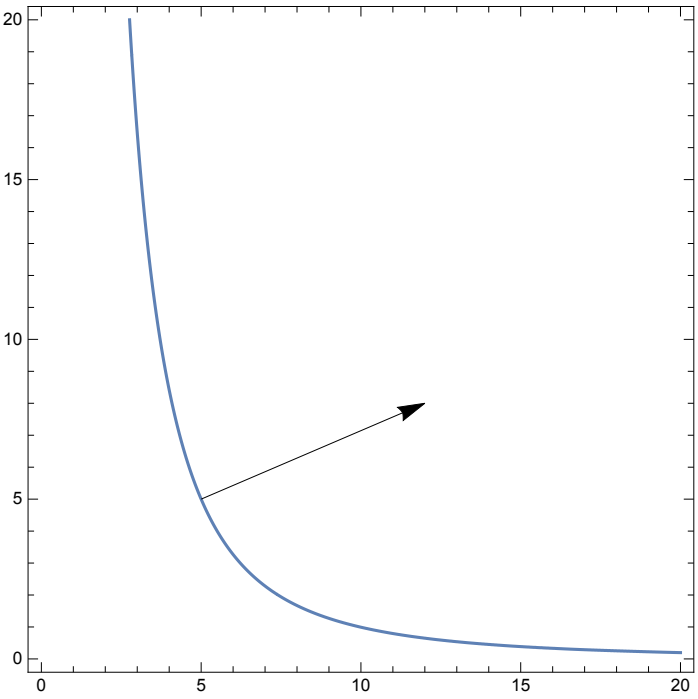
```
Function[{n, k}, 10 n^0.7 k^0.3]
```

$$\left\{\frac{7.\,k^{0.3}}{n^{0.3}},\ \frac{3.\,n^{0.7}}{k^{0.7}}\right\}$$

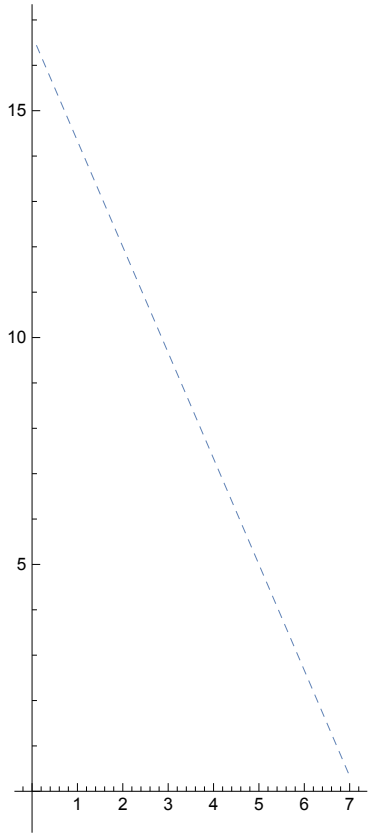
```
{n → 5, k → 5}
output at (5,5):
50.
the gradient at (5,5):
{7., 3.}
{5, 5, 50.}
{5 + dn, 5 + dk, 106.256}
```

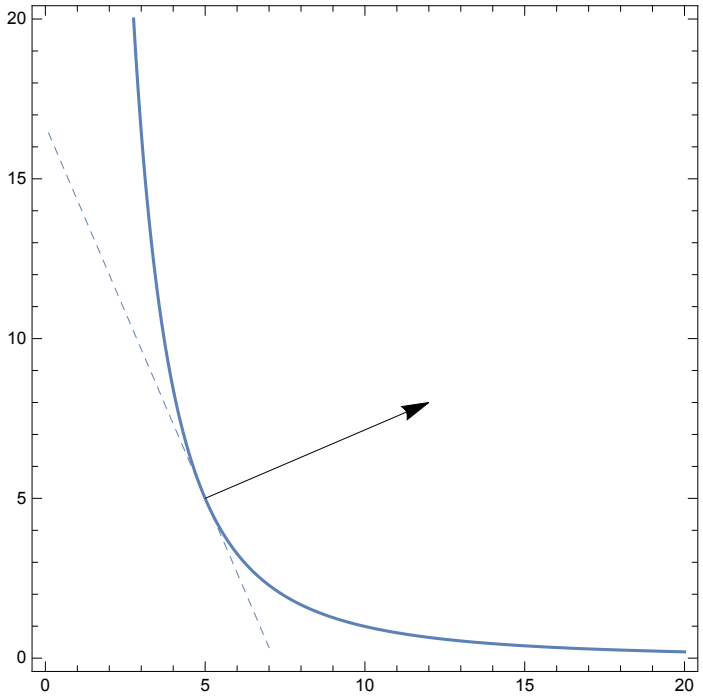


2d projection



-2.33333


15
10
5
1
2
3
4
5
6
7


20
15
10
5
0
0
5
10
15
20

Non-linear Comparative Statics

A Non-linear System of Equations

Consider the following as our "structual" equations (without worrying too much about the nature of economic structure).

e:islm.gmkt

$$Y = A(i - \pi, Y, G)$$
$$m = L(i, Y) \tag{10}$$

Here Y is total production in the economy, A is the function determining demand for that production, i is the nominal interest rate, π is the expected inflation rate, G measures the "fiscal stance" (i.e., how expansionary fiscal policy is), and m is the real money supply.

```
ClearAll[fA, fL, i, y, m, Π, g] (* all Mma variables should start lower case *)
lmnl = m == fL[i, y];
isnl = y == fA[i - Π, y, g];
```

Produce a textbook "Keynesian" model by taking Y and i to be endogenous, or produce a textbook "Classical" model by taking m and i to be endogenous.

Total Differentials

We will first consider the money market. Recall that equation (11) described money market equilibrium as m = L(i,Y) This must hold both before and after any exogenous changes. That is, we require that we start out in money market equilibrium, and we also require that we end up in money market equilibrium. It follows that the changes in the real money supply ($d\,m$) must equal the changes in real money demand ($d\,L$).

dm == dL (11)

The change in real money demand has two sources: changes in i and changes in Y. As usual, we will represent these as $d\,i$ and $d\,Y$. Of course, the change in real money demand depends not only on the size of the changes in these arguments, but also on how sensitive money demand is to each of these arguments.

dL = L_i di + L_Y dY (12)

Putting these two observations together, we get

e:islm.dmdl

dm = L_i di + L_Y dY (13)

We call this the "total differential" of the LM equation. It makes a very simple statement: we start out on an LM curve, and we end up on an LM curve.

```
dlmnl = Dt[lmnl]
```

```
Dt[m] == Dt[y] fL^(0,1)[i, y] + Dt[i] fL^(1,0)[i, y]
```

Let's make this a bit easier to read (but not to manipulate) by introducing some notation as rules. Note

that this is purely for convenience in reading: we are using strings (not symbols) in the result.

```
notationRulesLM =
  {fL^(0,1)[i, y] → "L_y", fL^(1,0)[i, y] → "L_i", Dt[m] → dm, Dt[i] → di, Dt[y] → dy};
dlmnl /. notationRulesLM
```

```
dm == L_i di + L_y dy
```

Next consider the goods market. Recall that the equation

$$Y = A(i - \pi, Y, G) \tag{14}$$

represents equilibrium in the goods market. This must hold both before and after any exogenous changes. That is, we require that we start out in goods market equilibrium, and we also require that we end up in goods market equilibrium. It follows that the changes in real income must equal the changes in real aggregate demand. Looking at the equation for the IS curve, we can see that this means that the change in real income (dY) must equal the change in real aggregate demand (dA).

$$dY == dA \tag{15}$$

The change in aggregate demand has three sources: changes in r, changes in Y, and changes in F. We represent these changes as dr, dY, and dG. Of course, the changes in aggregate demand depend not only on the size of the changes in these arguments, but also on how sensitive aggregate demand is to each of these arguments.

$$dA = A_r \overbrace{(di - d\Pi)}^{dr} + A_Y\, dY + A_G\, dG \tag{16}$$

Putting these two pieces together, we have the total differential of the IS equation:

$$dY = A_r(di - d\pi) + A_Y\, dY + A_G\, dG \tag{17}$$

Note that $A(\cdot, \cdot, \cdot)$ has only three arguments. Do not be misled by the fact that we choose to write r as $i - \pi$. This does not change the number of arguments of the aggregate demand function. E.g., there is no derivative A_i.

```
disnl = Dt[isnl]
```

```
Dt[y] == Dt[g] fA^(0,0,1)[i - Π, y, g] +
  Dt[y] fA^(0,1,0)[i - Π, y, g] + (Dt[i] - Dt[Π]) fA^(1,0,0)[i - Π, y, g]
```

```
notationRulesIS = {fA^(0,0,1)[i - Π, y, g] → "A_G",
   fA^(0,1,0)[i - Π, y, g] → "A_y", fA^(1,0,0)[i - Π, y, g] → "A_r", Dt[g] → dg, Dt[Π] → dΠ};
notationRules = Join[notationRulesLM, notationRulesIS];
disnl /. notationRules
```

```
dy == A_G dg + A_y dy + A_r (di - dΠ)
```

Implicit Function Theorem

The IFT provides the conditions under which we can characterize the partial derivatives of the reduced form in terms of the partial derivatives of the structural form. That is, we can do qualitative comparative statics.

Review the IFT using the online notes.

"Keynesian" Model

Let us first consider a textbook Keynesian model. Assuming satisfaction of the assumptions of the implicit function theorem, there is an implied reduced form for the Keynesian model. The reduced form expresses the solution for each endogenous variables in terms of the exogenous variables. We will represent this as

$$\begin{aligned} i &= i(m, \pi, G) \\ Y &= Y(m, \pi, G) \end{aligned} \tag{18}$$

The implicit function theorem tells us how to find the partial derivatives of $i(.,.)$ and $Y(.,.)$.

Note how we use the letter i to represent both a variable (on the left) and a function (on the right). This is common practice among economists and mathematicians, as it helps us keep track of which function is related to which variable. (However we will not usually be able to do this in a computer algebra system.) Note that since we did not begin with an explicit functional form for the structural equations we cannot hope to find an explicit functional form for the reduced form. Instead we rely on qualitative information about the structural equations to make qualitative statements about the reduced form.

```
dlmnl
```

```
dlmnl /. notationRules
```

```
Dt[m] == Dt[y] fL^(0,1)[i, y] + Dt[i] fL^(1,0)[i, y]
```

```
dm == L_i di + L_y dy
```

The total differential can be used to find the slope of the LM curve. Suppose we allow only i and Y to change (so that $dm = 0$). Then we must have

$$\begin{aligned} 0 &= L_i\, di + L_Y\, dY \\ \left.\frac{di}{dY}\right|_{LM} &= -\frac{L_Y}{L_i} > 0 \end{aligned} \tag{19}$$

```
Dt[lmnl] /. {Dt[m] → 0} /. notationRules
(* represent restricted total differential *)
Solve[(Dt[lmnl] /. {Dt[m] → 0, Dt[y] → 1}), Dt[i]] /. notationRules
```

```
0 == L_i di + L_y dy
```

$$\left\{\left\{di \to -\frac{L_y}{L_i}\right\}\right\}$$

This represents the way i and Y must change together to maintain equilibrium in the money market, ceteris paribus. That is, this determines the slope of the "Keynesian" LM curve. Under the standard assumptions that $L_Y > 0$ and $L_i < 0$, the "Keynesian" LM curve has a positive slope.

Similarly, if we allow only i and Y to change in the goods market, we must have

$$d\,Y = A_r\,d\,i + A_Y\,d\,Y$$
$$\left.\frac{d\,i}{d\,Y}\right|_{IS} = \frac{1 - A_Y}{A_r} < 0 \tag{20}$$

```
restrictions = {Dt[m] → 0, Dt[g] → 0, Dt[Π] → 0}
Dt[isnl] /. restrictions /. notationRules
(* represent restricted total differential *)
Solve[(Dt[isnl] /. restrictions /. {Dt[y] → 1}), Dt[i]] /. notationRules
```

```
{Dt[m] → 0, Dt[g] → 0, Dt[Π] → 0}
```

```
dy == Ar di + Ay dy
```

$$\left\{\left\{\text{di} \to \frac{1 - \text{Ay}}{\text{Ar}}\right\}\right\}$$

This is the way i and Y must change together to maintain equilibrium in the goods market. That is, this determines the slope of the "Keynesian" IS curve. Under the standard assumptions that 0 < A_Y < 1 and A_r < 0, the "Keynesian" IS curve has a negative slope.

Solving the Nonlinear Keynesian Model

So we have seen what is required to stay on the IS curve and what is required to stay on the LM curve. Putting these together we have

$$\begin{aligned} d\,Y &= A_r(\text{di} - d\pi) + A_Y\,d\,Y + A_F\,dG \\ dm &= L_i\,\text{di} + L_Y\,dY \end{aligned} \tag{21}$$

When we insist that both of these equation hold together, we are insisting that we stay on both the IS and LM curves simultaneously. In this system there are two endogenous variables, $d\,r$ and $d\,Y$, which are being determined so as to achieve this simultaneous satisfaction of the IS and LM equations.

Now we just solve two linear equations in two unknowns. First prepare to set up the system as a matrix equation by moving all terms involving the endogenous variables to the left. (Note that this is the first time we have paid attention to which variables are endogenous.)

$$\begin{aligned} -A_r\,\text{di} + dY - A_Y\,\text{dY} &= -A_r\,d\,\pi + A_F\,dG \\ L_i\,\text{di} + L_Y\,\text{dY} &= dm \end{aligned} \tag{22}$$

Now rewrite this system as a matrix equation in the form $J\,x = b$.

$$\begin{bmatrix} -A_r & (1 - A_Y) \\ L_i & L_Y \end{bmatrix}\begin{bmatrix} di \\ dY \end{bmatrix} = \begin{bmatrix} -A_r\,d\,\pi + A_F\,\text{dG} \\ dm \end{bmatrix} \tag{23}$$

Then solve for the endogenous variables by multiplying both sides by J^{-1}.

$$\begin{aligned} \begin{bmatrix} d\,i \\ d\,Y \end{bmatrix} &= \frac{1}{-A_r\,L_Y - (1 - A_Y)\,L_i}\begin{bmatrix} L_Y & -(1 - A_Y) \\ -L_i & -A_r \end{bmatrix}\begin{bmatrix} -A_r\,d\,\pi + A_F\,\text{dG} \\ d\,m \end{bmatrix} \\ &= \frac{1}{A_r\,L_Y + (1 - A_Y)\,L_i}\begin{bmatrix} -L_Y & (1 - A_Y) \\ L_i & A_r \end{bmatrix}\begin{bmatrix} -A_r\,d\,\pi + A_F\,\text{dG} \\ d\,m \end{bmatrix} \end{aligned} \tag{24}$$

Letting $\Delta = A_r L_Y + (1 - A_Y) L_i$, we can write this as

$$\begin{bmatrix} d\,i \\ d\,Y \end{bmatrix} = \frac{1}{\Delta}\begin{bmatrix} -L_Y & (1 - A_Y) \\ L_i & A_r \end{bmatrix}\begin{bmatrix} -A_r\,d\,\pi + A_G\,dG \\ d\,m \end{bmatrix} \tag{25}$$

Invoking the standard assumptions on the structural form partial derivatives, listed above, we note that $\Delta = A_r L_Y + (1 - A_Y) L_i < 0$.

```
solnK = Solve[dlmnl && disnl, {Dt[i], Dt[y]}];
solnK /. notationRules
```

$$\left\{\left\{\text{di} \to -\frac{-\text{A}_\text{G}\,\text{L}_\text{y}\,\text{dg} + \text{dm} - \text{A}_\text{y}\,\text{dm} + \text{A}_\text{r}\,\text{L}_\text{y}\,\text{d}\Pi}{-\text{L}_\text{i} + \text{A}_\text{y}\,\text{L}_\text{i} - \text{A}_\text{r}\,\text{L}_\text{y}},\ \text{dy} \to -\frac{-\text{A}_\text{G}\,\text{L}_\text{i}\,\text{dg} - \text{A}_\text{r}\,\text{dm} + \text{A}_\text{r}\,\text{L}_\text{i}\,\text{d}\Pi}{\text{L}_\text{i} - \text{A}_\text{y}\,\text{L}_\text{i} + \text{A}_\text{r}\,\text{L}_\text{y}}\right\}\right\}$$

Fiscal policy experiment:

$$\begin{aligned}\begin{bmatrix} \partial i/\partial G \\ \partial Y/\partial G \end{bmatrix} &= \frac{1}{\Delta}\begin{bmatrix} -L_Y & (1 - A_Y) \\ L_i & A_r \end{bmatrix}\begin{bmatrix} A_G \\ 0 \end{bmatrix} \\ &= \frac{1}{\Delta}\begin{bmatrix} -L_Y A_F \\ L_i A_F \end{bmatrix} = \begin{bmatrix} + \\ + \end{bmatrix}\end{aligned} \tag{26}$$

```
(* find the partial responses to dg *)
$Assumptions = 1 > fA^(0,1,0)[i - Π, y, g] > 0 && fA^(0,0,1)[i - Π, y, g] > 0 &&
  fA^(1,0,0)[i - Π, y, g] < 0 && fL^(0,1)[i, y] > 0 && fL^(1,0)[i, y] < 0
gpartials = solnK /. {Dt[m] → 0, Dt[Π] → 0, Dt[g] → 1}
gpartials /. notationRules // Simplify
{didg, dydg} = {Dt[i], Dt[y]} /. gpartials[[1]]
Sign[{didg, dydg}] // Simplify
```

```
1 > fA^(0,1,0)[i - Π, y, g] > 0 && fA^(0,0,1)[i - Π, y, g] > 0 &&
 fA^(1,0,0)[i - Π, y, g] < 0 && fL^(0,1)[i, y] > 0 && fL^(1,0)[i, y] < 0
```

$$\left\{\left\{\text{Dt[i]} \to \left(\text{fL}^{(0,1)}[\text{i, y}]\,\text{fA}^{(0,0,1)}[\text{i} - \Pi, \text{y, g}]\right) \Big/ \left(-\text{fL}^{(1,0)}[\text{i, y}] + \text{fL}^{(1,0)}[\text{i, y}]\,\text{fA}^{(0,1,0)}[\text{i} - \Pi, \text{y, g}] - \text{fL}^{(0,1)}[\text{i, y}]\,\text{fA}^{(1,0,0)}[\text{i} - \Pi, \text{y, g}]\right),\ \text{Dt[y]} \to \left(\text{fL}^{(1,0)}[\text{i, y}]\,\text{fA}^{(0,0,1)}[\text{i} - \Pi, \text{y, g}]\right) \Big/ \left(\text{fL}^{(1,0)}[\text{i, y}] - \text{fL}^{(1,0)}[\text{i, y}]\,\text{fA}^{(0,1,0)}[\text{i} - \Pi, \text{y, g}] + \text{fL}^{(0,1)}[\text{i, y}]\,\text{fA}^{(1,0,0)}[\text{i} - \Pi, \text{y, g}]\right)\right\}\right\}$$

$$\left\{\left\{\text{di} \to -\frac{\text{A}_\text{G}\,\text{L}_\text{y}}{\text{L}_\text{i} - \text{A}_\text{y}\,\text{L}_\text{i} + \text{A}_\text{r}\,\text{L}_\text{y}},\ \text{dy} \to \frac{\text{A}_\text{G}\,\text{L}_\text{i}}{\text{L}_\text{i} - \text{A}_\text{y}\,\text{L}_\text{i} + \text{A}_\text{r}\,\text{L}_\text{y}}\right\}\right\}$$

$$\left\{\left(\text{fL}^{(0,1)}[\text{i, y}]\,\text{fA}^{(0,0,1)}[\text{i} - \Pi, \text{y, g}]\right) \Big/ \left(-\text{fL}^{(1,0)}[\text{i, y}] + \text{fL}^{(1,0)}[\text{i, y}]\,\text{fA}^{(0,1,0)}[\text{i} - \Pi, \text{y, g}] - \text{fL}^{(0,1)}[\text{i, y}]\,\text{fA}^{(1,0,0)}[\text{i} - \Pi, \text{y, g}]\right),\ \left(\text{fL}^{(1,0)}[\text{i, y}]\,\text{fA}^{(0,0,1)}[\text{i} - \Pi, \text{y, g}]\right) \Big/ \left(\text{fL}^{(1,0)}[\text{i, y}] - \text{fL}^{(1,0)}[\text{i, y}]\,\text{fA}^{(0,1,0)}[\text{i} - \Pi, \text{y, g}] + \text{fL}^{(0,1)}[\text{i, y}]\,\text{fA}^{(1,0,0)}[\text{i} - \Pi, \text{y, g}]\right)\right\}$$

```
{1, 1}
```

Monetary policy experiment:

We know from the implicit function theorem that this is the same as solving for the partial derivatives of

the reduced form. , we can write

$$\begin{bmatrix} \partial i/\partial m \\ \partial Y/\partial m \end{bmatrix} = \frac{1}{\Delta}\begin{bmatrix} -L_Y & (1-A_Y) \\ L_i & A_r \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\Delta}\begin{bmatrix} (1-A_Y) \\ A_r \end{bmatrix} = \begin{bmatrix} - \\ + \end{bmatrix} \tag{27}$$

```
(* find the partial responses to dm *)
mpartials = solnK /. {Dt[g] → 0, Dt[Π] → 0, Dt[m] → 1};
mpartials /. notationRules // Simplify
{didm, dydm} = {Dt[i], Dt[y]} /. mpartials[[1]]
Sign[{didm, dydm}] // Simplify
```

$$\left\{\left\{\text{di} \to \frac{1 - \text{A}_\text{y}}{\text{L}_\text{i} - \text{A}_\text{y}\,\text{L}_\text{i} + \text{A}_\text{r}\,\text{L}_\text{y}},\ \text{dy} \to \frac{\text{A}_\text{r}}{\text{L}_\text{i} - \text{A}_\text{y}\,\text{L}_\text{i} + \text{A}_\text{r}\,\text{L}_\text{y}}\right\}\right\}$$

$$\left\{-\left(\left(1 - \text{fA}^{(0,1,0)}[\text{i} - \Pi, \text{y}, \text{g}]\right) \Big/ \left(-\text{fL}^{(1,0)}[\text{i}, \text{y}] + \text{fL}^{(1,0)}[\text{i}, \text{y}]\ \text{fA}^{(0,1,0)}[\text{i} - \Pi, \text{y}, \text{g}] - \text{fL}^{(0,1)}[\text{i}, \text{y}]\ \text{fA}^{(1,0,0)}[\text{i} - \Pi, \text{y}, \text{g}]\right)\right),\ \text{fA}^{(1,0,0)}[\text{i} - \Pi, \text{y}, \text{g}] \Big/ \left(\text{fL}^{(1,0)}[\text{i}, \text{y}] - \text{fL}^{(1,0)}[\text{i}, \text{y}]\ \text{fA}^{(0,1,0)}[\text{i} - \Pi, \text{y}, \text{g}] + \text{fL}^{(0,1)}[\text{i}, \text{y}]\ \text{fA}^{(1,0,0)}[\text{i} - \Pi, \text{y}, \text{g}]\right)\right\}$$

```
{-1, 1}
```

Experiment: change in expected inflation.

Recall our reduce form:

$$\begin{bmatrix} d\,i \\ d\,Y \end{bmatrix} = \frac{1}{\Delta}\begin{bmatrix} -L_Y & (1-A_Y) \\ L_i & A_r \end{bmatrix}\begin{bmatrix} -A_r\,d\,\pi + A_G\,dG \\ d\,m \end{bmatrix} \tag{28}$$

Now set dG = 0 and dm = 0.

$$\begin{bmatrix} d\,i \\ d\,Y \end{bmatrix} = \frac{1}{\Delta}\begin{bmatrix} -L_Y & (1-A_Y) \\ L_i & A_r \end{bmatrix}\begin{bmatrix} -A_r\,d\pi \\ 0 \end{bmatrix} \tag{29}$$

Now divide both sides by dπ.

$$\begin{bmatrix} \partial i/\partial \pi \\ \partial Y/\partial \pi \end{bmatrix} = \frac{1}{\Delta}\begin{bmatrix} -L_Y & (1-A_Y) \\ L_i & A_r \end{bmatrix}\begin{bmatrix} -A_r \\ 0 \end{bmatrix} = \frac{1}{\Delta}\begin{bmatrix} L_Y A_r \\ -L_i A_r \end{bmatrix} = \begin{bmatrix} + \\ + \end{bmatrix} \tag{30}$$

“Classical” Model (Exercise)

In the Classical case we follow the same procedures and the same type of reasoning, making only a single change: instead of Y we take m to be endogenous, so that m and i are the endogenous variables. Note that we start with the *same* system of structural equations:

$$Y = A(i - \pi, Y, F)$$
$$m = L(i, Y) \tag{31}$$

It follows that the total differential is unchanged:

$$dY = A_r(di - d\pi) + A_Y dY + A_F dF$$
$$dm = L_i di + L_Y dY \tag{32}$$

Of course, all the partial derivatives from the structural form are unchanged: $A_r < 0$, $0 < A_Y < 1$, $A_F > 0$, $L_i < 0$, and $L_Y > 0$.

But of course we have a different set of endogenous variables, so we have a different implied reduced form:

$$m = m(Y, \pi, F)$$
$$i = i(Y, \pi, F) \tag{33}$$

So when we write down the matrix equation, we use our new set of endogenous variables:

$$\begin{bmatrix} A_r & 0 \\ -L_i & 1 \end{bmatrix}\begin{bmatrix} di \\ dm \end{bmatrix} = \begin{bmatrix} A_r d\pi + (1 - A_Y) dY - A_F dF \\ L_Y dY \end{bmatrix} \tag{34}$$

Solving for the changes in the endogenous variables:

$$\begin{bmatrix} di \\ dm \end{bmatrix} = \frac{1}{A_r}\begin{bmatrix} 1 & 0 \\ L_i & A_r \end{bmatrix}\begin{bmatrix} A_r d\pi + (1 - A_Y) dY - A_F dF \\ L_Y dY \end{bmatrix} \tag{35}$$

So for example

$$\begin{aligned} \begin{bmatrix} \partial i/\partial \pi \\ \partial m/\partial \pi \end{bmatrix} &= \frac{1}{A_r}\begin{bmatrix} 1 & 0 \\ L_i & A_r \end{bmatrix}\begin{bmatrix} A_r \\ 0 \end{bmatrix} \\ &= \frac{1}{A_r}\begin{bmatrix} A_r \\ L_i A_r \end{bmatrix} \\ &= \begin{bmatrix} + \\ - \end{bmatrix} \end{aligned} \tag{36}$$

Optimization

Univariate Optimization

For optimization, you may want to use `Maximize` or `FindMaximum`.

Profit Maximization Example

```
Clear[q, p]
demand = q == 1500 - p / 2
inverseDemand = Solve[demand, p] // Simplify
tr = p * q /. First[inverseDemand] (* total revenue *)
```

$$q == 1500 - \frac{p}{2}$$

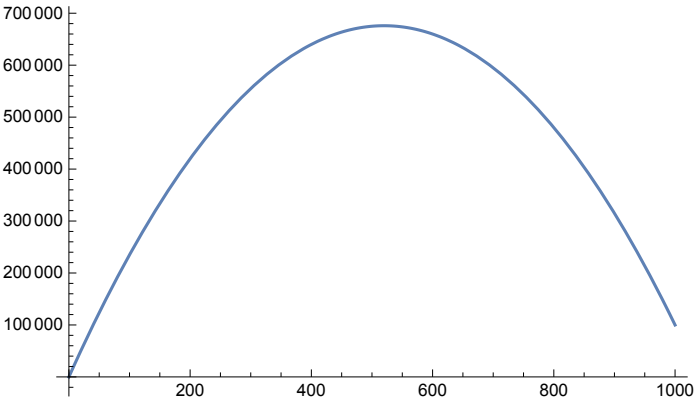
$$\{\{p \to 3000 - 2\,q\}\}$$

$$(3000 - 2\,q)\,q$$

```
(* specify total cost fn *)
tc = 50 + 400 * q + q^2 / 2;
profits = tr - tc // Simplify
```

$$-\frac{5}{2}\left(20 - 1040\,q + q^2\right)$$

```
Plot[profits, {q, 0, 1000}]
```




```
dπdq = D[profits, q]
foc = dπdq == 0
```

$$-\frac{5}{2}\,(-1040 + 2\,q)$$

$$-\frac{5}{2}\,(-1040 + 2\,q) == 0$$

```
Solve[foc, q]
proposedMax = First[%]
```

$$\{\{q \to 520\}\}$$

$$\{q \to 520\}$$

Check the curvature at the stationary point.

```
D[profits, {q, 2}]
```

```
-5
```

When we are interested in global extrema, we can use Maximize or Minimize. These work especially well with polynomials.

```
Maximize[profits, q]
```

```
{675 950, {q → 520}}
```

```
Maximize[{E^(-0.05*t) * E^Sqrt[t/2], t > 0}, t]
```

```
{12.1825, {t → 50.}}
```

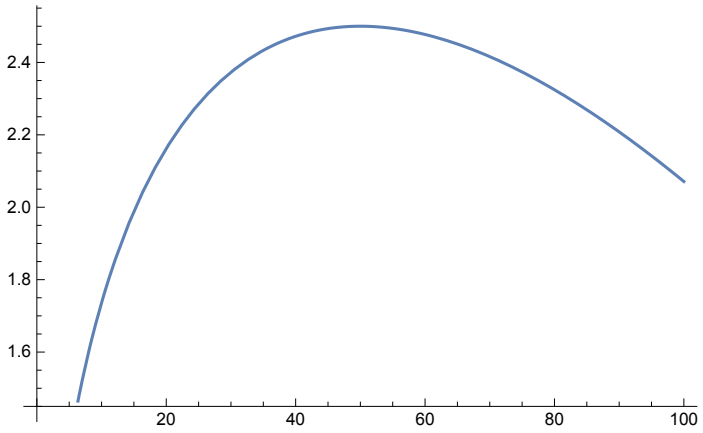
```
Clear[t]
f = t ↦ Sqrt[t/2] - 0.05 * t
Maximize[{Sqrt[t/2] - 0.05 * t, t > 0}, t]
```

```
Function[t, Sqrt[t/2] - 0.05 t]
```

```
{2.5, {t → 50.}}
```

```
f = t ↦ Sqrt[t/2] - 0.05 * t
Plot[f[t], {t, 0, 100}]
```

```
Function[t, Sqrt[t/2] - 0.05 t]
```



```
f'[t]
foc = f'[t] == 0
Solve[foc, t]
f''[t]
```

$$-0.05 + \frac{1}{2\sqrt{2}\sqrt{t}}$$

$$-0.05 + \frac{1}{2\sqrt{2}\sqrt{t}} == 0$$

$$\{\{t \to 50.\}\}$$

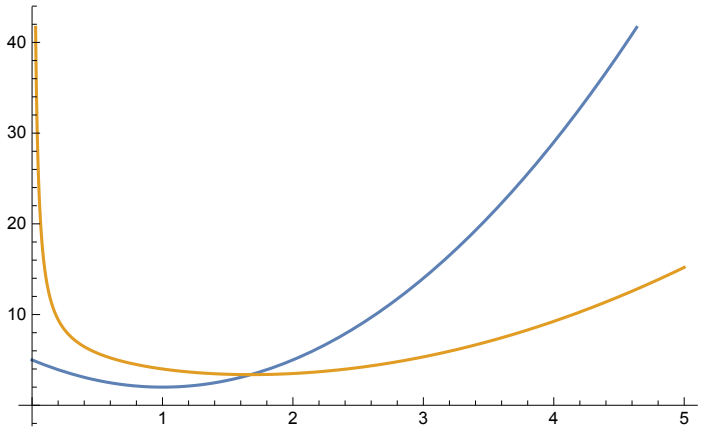
$$-\frac{1}{4\sqrt{2}\,t^{3/2}}$$

Cost Functions

```
tc = q^3 - 3 * q^2 + 5 * q + 1;
{"marginal cost", mc = D[tc, q]}
{"average cost", ac = tc / q} // Apart
Plot[{mc, ac}, {q, 0, 5}]
```

$$\{\text{marginal cost}, 5 - 6\,q + 3\,q^2\}$$

$$\{\text{average cost}, 5 + \frac{1}{q} - 3\,q + q^2\}$$




```
NSolve[mc == ac, q, Reals]
```

$$\{\{q \to 1.67765\}\}$$

```
NSolve[D[ac, q] == 0 && q > 0, q]
NMinimize[{ac, q > 0}, q]
```

$$\{\{q \to 1.67765\}\}$$

$$\{3.37763, \{q \to 1.67765\}\}$$

Basic Considerations

```
ClearAll["Global`*"]
```

When we are interested in global extrema, we can use Maximize or Minimize. (To search locally use FindMaximum and FindMinimum.) These work especially well with polynomials, but can work with many functions.

```
Maximize[{E^(-0.05*t) * E^Sqrt[t/2], t > 0}, t]
```

```
{12.1825, {t → 50.}}
```

A strictly increasing tranformation of the objective function produces the same maximizer. Of course, the value of the objective function will be transformed.

```
Maximize[{Sqrt[t/2] - 0.05 * t, t > 0}, t]
```

```
{2.5, {t → 50.}}
```

While the use of Maximize is convenient, we whould also be able to approach the problem from first principles.

```
f2max = Sqrt[t/2] - 0.05 * t;
tmax = Solve[D[f2max, t] == 0, t] // Simplify
f2max /. tmax
```

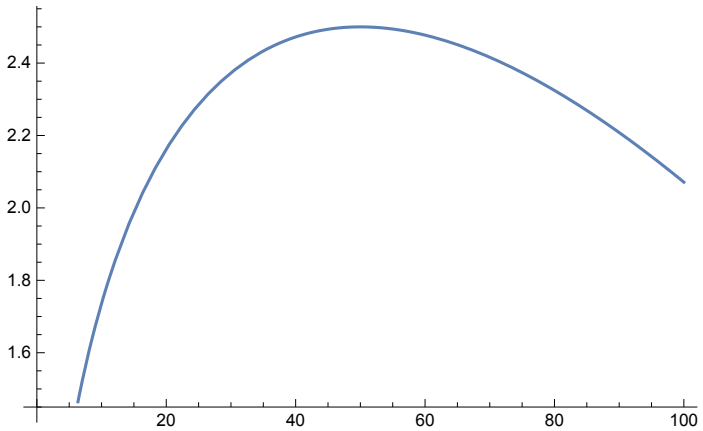
```
{{t → 50.}}
```

```
{2.5}
```

Of course we should consider the curvature of the function as well. We can check that the second derivative is negative, or we can just look at a plot:

```
D[f2max, {t, 2}] < 0 /. tmax // Simplify
Plot[f2max, {t, 0, 100}]
```

```
{True}
```



Of course we can do the same exercise with an explicit function definition.

```
f[t_] := √(t/2) - 0.05 * t
tmax = Solve[f'[t] == 0, t]
f''[t] < 0 /. tmax // Simplify
```

```
{{t → 50.}}
```

```
{True}
```

Laffer Curve

```
taxes = θ (t - t^2)
dtaxes = D[taxes, t]
Solve[dtaxes == 0, t]
d2taxes = D[dtaxes, t] (* check second order condition *)
```

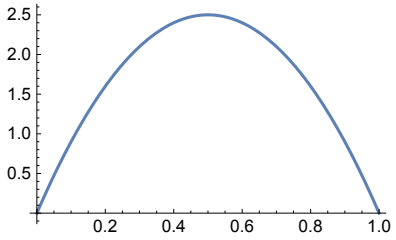
$$(t - t^2)\,\theta$$

$$(1 - 2\,t)\,\theta$$

$$\left\{\left\{t \to \frac{1}{2}\right\}\right\}$$

$$-2\,\theta$$

```
Plot[taxes /. {θ → 10}, {t, 0, 1}, ImageSize → 200]
```



Minimizing a Loss Function

```
L = -0.5 * (pi - pie) + pi^2
dL = D[L, pi]
Solve[dL == 0, pi]
D[dL, pi]
```

$$pi^2 - 0.5\,(pi - pie)$$

$$-0.5 + 2\,pi$$

```
{{pi → 0.25}}
```

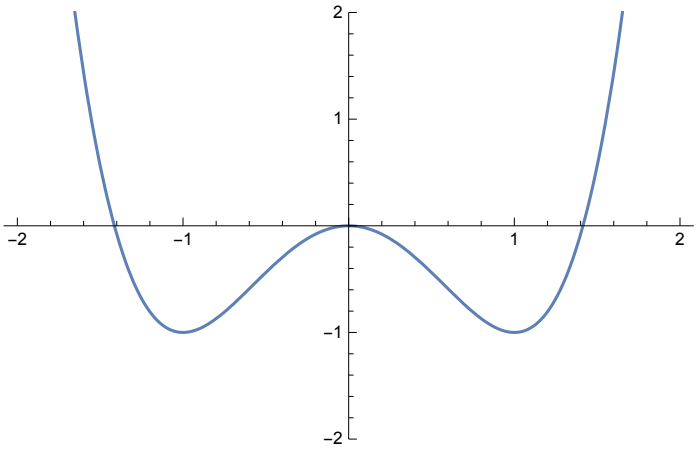
```
2
```

```
pi^2 - 0.5 * (pi - pie)
```

$$pi^2 - 0.5\,(pi - pie)$$

```
Clear[x, y]
y = x^4 - 2 * x^2
Plot[y, {x, -2, 2}, PlotRange → {-2, 2}]
```

$$-2\,x^2 + x^4$$

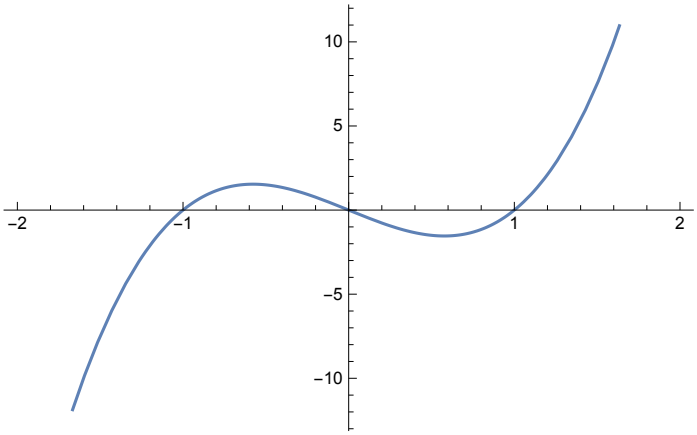



```
y
dydx = D[y, x]
```

$$-2\,x^2 + x^4$$

$$-4\,x + 4\,x^3$$

```
Plot[dydx, {x, -2, 2}]
```




```
soln05 = Solve[dydx == 0, x]
```

$$\{\{x \to -1\}, \{x \to 0\}, \{x \to 1\}\}$$

```
D[dydx, x] /. soln05
```

$$\{8, -4, 8\}$$

Profit Maximizing Monopolist

Let us apply this to a profit maximizing monopolist.

```
ClearAll["Global`*"]
"assumed demand schedule"
demand = q == 1500 - p / 2
"inverse demand:"
inverseDemand = Solve[demand, p][[1]] // Simplify
"total revenue:"
totalRevenue = p * q /. inverseDemand
```

```
assumed demand schedule
```

$$q == 1500 - \frac{p}{2}$$

```
inverse demand:
```

$$\{p \to 3000 - 2\,q\}$$

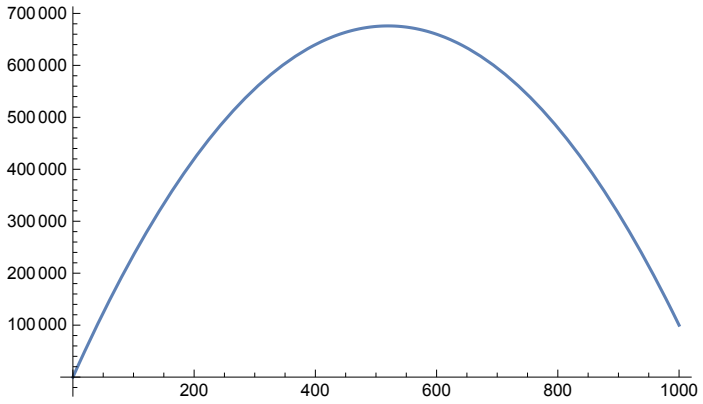
```
total revenue:
```

$$(3000 - 2\,q)\,q$$

No introduce some costs of production:

```
totalCost = 50 + 400 * q + q^2 / 2;
profits = totalRevenue - totalCost // Simplify
Plot[profits, {q, 0, 1000}]
```

$$-\frac{5}{2}\left(20 - 1040\,q + q^2\right)$$



Once again, we can take advantage of the Maximize command:

```
Maximize[profits, q]
```

$$\{675\,950, \{q \to 520\}\}$$

Or we can approach this from first principles:

```
dprofits = D[profits, q]
foc = dprofits == 0
soln = Solve[foc, q]
profits /. soln
```

$$-\frac{5}{2}\,(-1040+2\,q)$$

$$-\frac{5}{2}\,(-1040+2\,q)==0$$

$$\{\{q\to 520\}\}$$

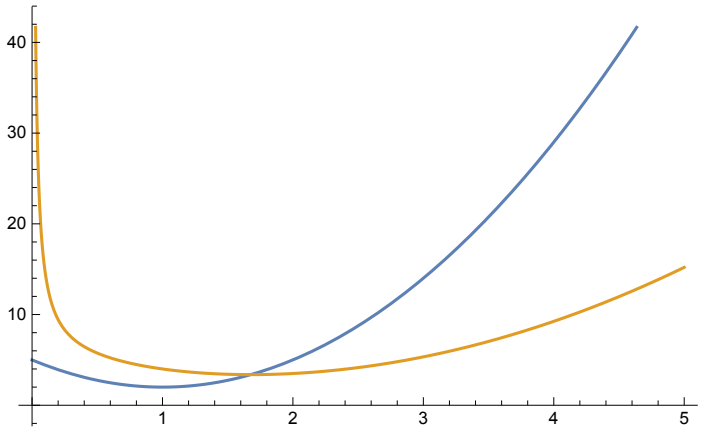
$$\{675\,950\}$$

Marginal Cost and Average Cost

```
tc = q^3 - 3 * q^2 + 5 * q + 1;
{"marginal cost", mc = D[tc, q]}
{"average cost", ac = tc / q}
Plot[{mc, ac}, {q, 0, 5}]
```

$$\{\text{marginal cost},\ 5-6\,q+3\,q^2\}$$

$$\left\{\text{average cost},\ \frac{1+5\,q-3\,q^2+q^3}{q}\right\}$$




```
mc == ac // Simplify
q`intersect = Solve[mc == ac, q, Reals]
```

$$2\,q^2==\frac{1}{q}+3\,q$$

$$\left\{\left\{q\to \text{Root}\left[-1-3\,\#1^2+2\,\#1^3\ \&,\ 1\right]\right\}\right\}$$

We can request an exact expression for this.

```
ToRadicals[q`intersect]
```

$$\left\{\left\{q \to \frac{1}{6}\left(3+\left(81-54\sqrt{2}\right)^{1/3}+3\left(3+2\sqrt{2}\right)^{1/3}\right)\right\}\right\}$$

Alternatively, we can request a numerical value.

```
N[q`intersect]
```

```
{{q → 1.67765}}
```

If we want a numerical value, we can more simply start out by requesting a numerical solution.

```
NSolve[mc == ac, q, Reals]
```

```
{{q → 1.67765}}
```

Next, let us consider the minimum of the average cost curve. It looks like average cost reaches a minimum where the two curves intersect. Let us check. We can use the Minimize command, as long as we are careful to impose the positivity constraint on q.

```
Minimize[{ac, q ≥ 0}, q] // Simplify
N[%]
```

$$\left\{5-3\,\text{Root}\left[-4-3\,\#1^2+\#1^3\ \&,\ 1\right]+\frac{3}{4}\,\text{Root}\left[-4-3\,\#1^2+\#1^3\ \&,\ 1\right]^2,\ \left\{q \to \text{Root}\left[-1-3\,\#1^2+2\,\#1^3\ \&,\ 1\right]\right\}\right\}$$

```
{3.37763, {q → 1.67765}}
```

Let us try this again, this time by being explicit about the first order condition.

```
q`minac = Solve[D[ac, q] == 0, Reals]
q`minac == q`intersect
```

$$\left\{\left\{q \to \text{Root}\left[-1-3\,\#1^2+2\,\#1^3\ \&,\ 1\right]\right\}\right\}$$

```
True
```

Two more times, this time using NSolve and NMinimize.

```
NSolve[D[ac, q] == 0 && q > 0, q]
NMinimize[{ac, q > 0}, q]
```

```
{{q → 1.67765}}
```

```
{3.37763, {q → 1.67765}}
```

Univariate Example: Laffer Curve

```
taxes = θ (t - t^2)
dtaxes = D[taxes, t]
Solve[dtaxes == 0, t]
d2taxes = D[dtaxes, t]
```

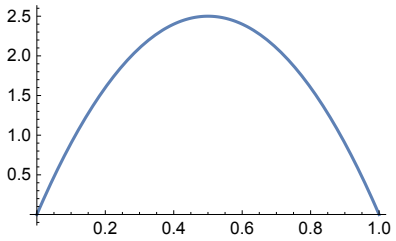
$$(t - t^2)\,\theta$$

$$(1 - 2\,t)\,\theta$$

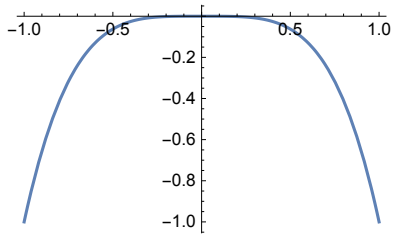
$$\left\{\left\{t \to \frac{1}{2}\right\}\right\}$$

$$-2\,\theta$$

```
Plot[taxes /. {θ → 10}, {t, 0, 1}, ImageSize → 200]
```




```
Plot[-x^4, {x, -1, 1}, ImageSize → 200]
```




```
L = -0.5 * (pi - pie) + pi^2
dL = D[L, pi]
Solve[dL == 0, pi]
D[dL, pi]
```

$$pi^2 - 0.5\,(pi - pie)$$

$$-0.5 + 2\,pi$$

$$\{\{pi \to 0.25\}\}$$

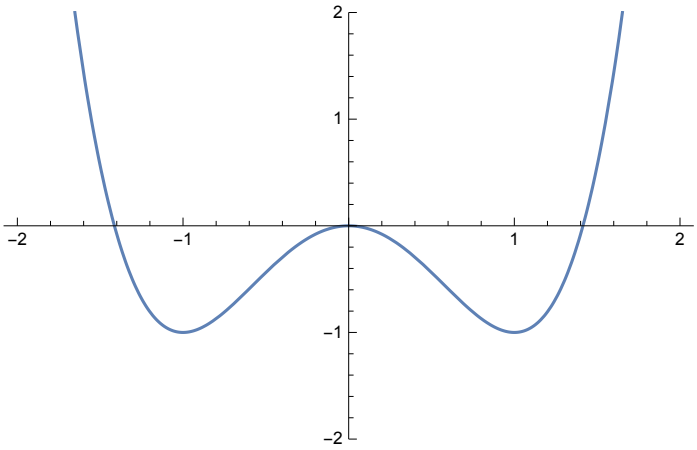
$$2$$

```
pi^2 - 0.5 * (pi - pie)
```

$$pi^2 - 0.5\,(pi - pie)$$

```
Clear[x, y]
y = x^4 - 2 * x^2
Plot[y, {x, -2, 2}, PlotRange → {-2, 2}]
```

$-2\,x^2 + x^4$

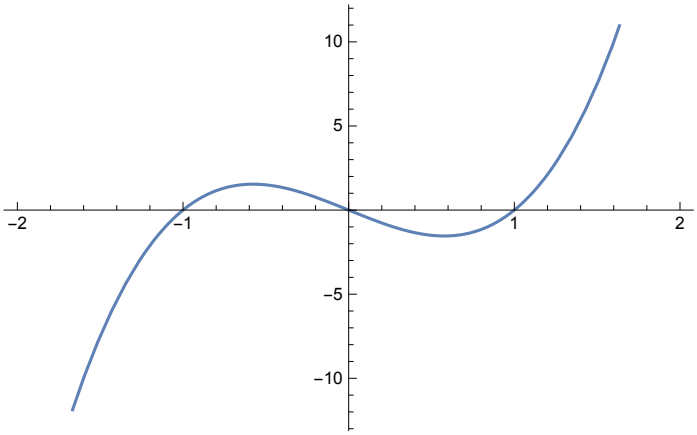



```
y
dydx = D[y, x]
```

$-2\,x^2 + x^4$

$-4\,x + 4\,x^3$

```
Plot[dydx, {x, -2, 2}]
```




```
soln05 = Solve[dydx == 0, x]
```

$\{\{x \to -1\}, \{x \to 0\}, \{x \to 1\}\}$

```
D[dydx, x] /. soln05
```

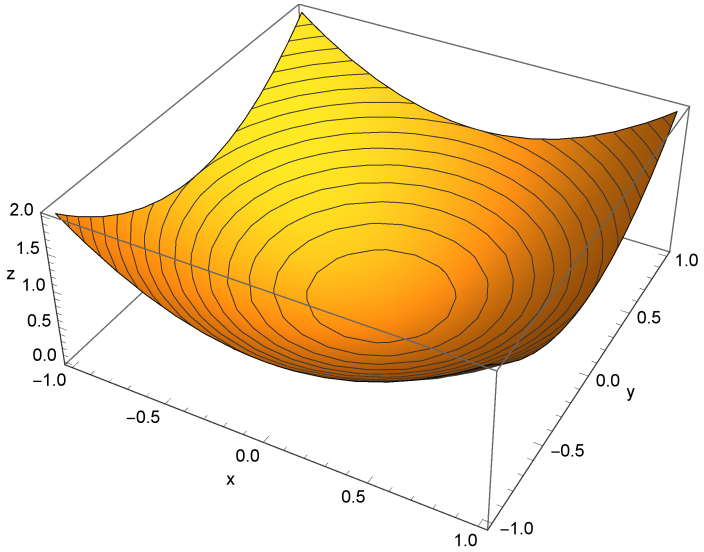
$\{8, -4, 8\}$

Bivariate Optimization

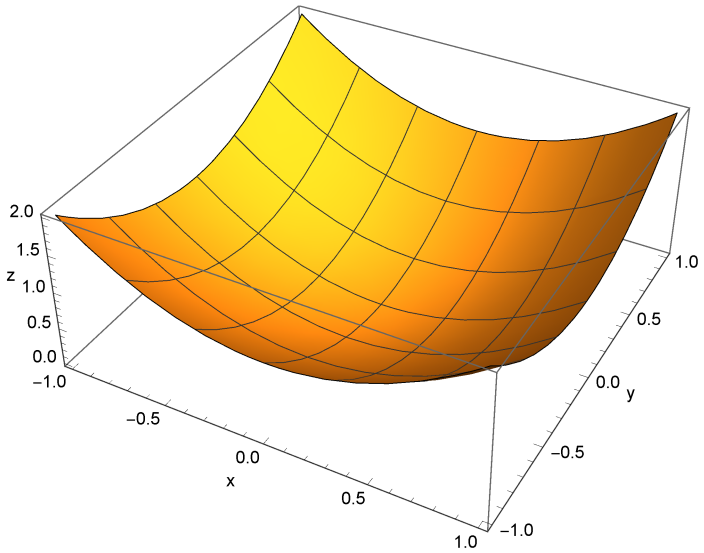
Basic Bivariate Example

```
Clear[x, y, z]
z = x^2 + y^2
Plot3D[z, {x, -1, 1}, {y, -1, 1},
 AxesLabel → {"x", "y", "z"}, MeshFunctions → {#3 &}]
```

$x^2 + y^2$

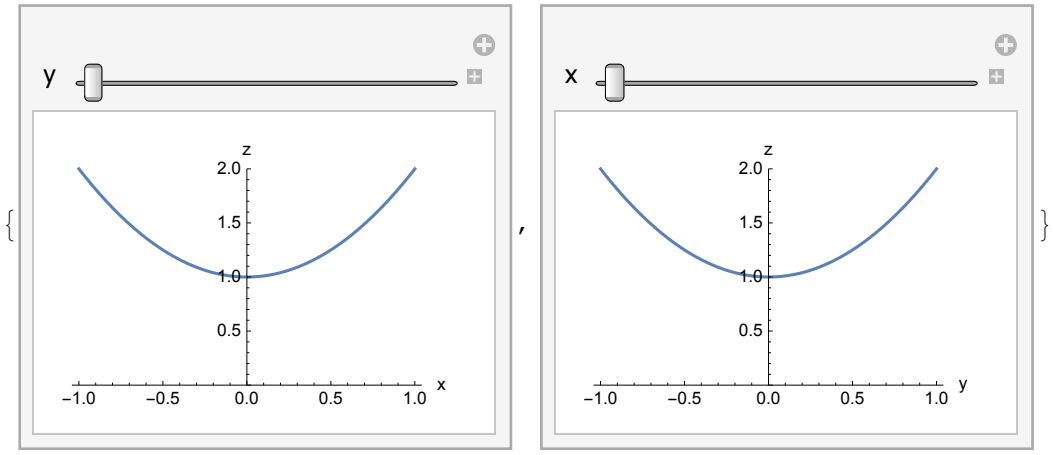



```
Plot3D[z, {x, -1, 1}, {y, -1, 1},
 AxesLabel → {"x", "y", "z"}, Mesh → 5]
```



If we hold one variable constant while varying the other, we produce two-dimensional plots. The slopes we see are partial derivatives.

```
{Manipulate[
  Plot[x^2 + y^2, {x, -1, 1}, PlotRange → {0, 2},
   AxesLabel → {"x", "z"}, ImageSize → 200], {y, -1, 1}],
 Manipulate[Plot[x^2 + y^2, {y, -1, 1}, PlotRange → {0, 2},
   AxesLabel → {"y", "z"}, ImageSize → 200], {x, -1, 1}]}
```



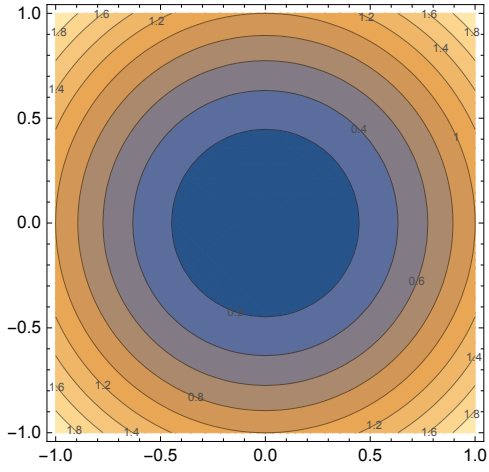
We search for an extremum looking at the stationary points: find (x_1, x_2) such that the two first-order partial derivatives equal zero. The slopes of the following grid lines are the partial derivatives.

```
gradient01 = D[z, {{x, y}}]
Solve[gradient01 == 0, {x, y}, Reals]
```

```
{2 x, 2 y}
```

```
{{x → 0, y → 0}}
```

```
ContourPlot[z, {x, -1, 1}, {y, -1, 1},
 ContourLabels → (Style[Text[#3, {#1, #2}], GrayLevel[.3], 6] &), ImageSize → 250]
```



Small Changes Can Matter

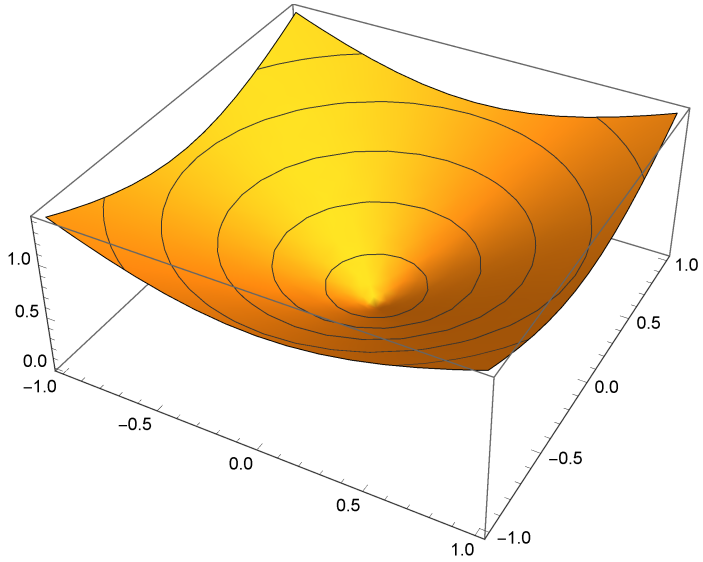
Suppose we are looking for an extremum of $f(x, y)$. If we consider any strictly increasing transformation

of *f*, it will have the same extrema.

```
d0 = √(x^2 + y^2)
```

```
Plot3D[d0, {x, -1, 1}, {y, -1, 1}, Mesh → 5, MeshFunctions → {#3 &}]
```

$$\sqrt{x^2 + y^2}$$



Again, search for an extremum by trying to find (x_1, x_2) such that the two first-order partial derivatives equal zero. What happens and why?

$$\frac{d}{dx}\sqrt{x^2+y^2} = \frac{d\sqrt{z}}{dz}\frac{dz}{dx} = \frac{1}{2\sqrt{z}}\frac{dz}{dx} = \frac{2x}{2\sqrt{x^2+y^2}}$$

where $z = x^2 + y^2$

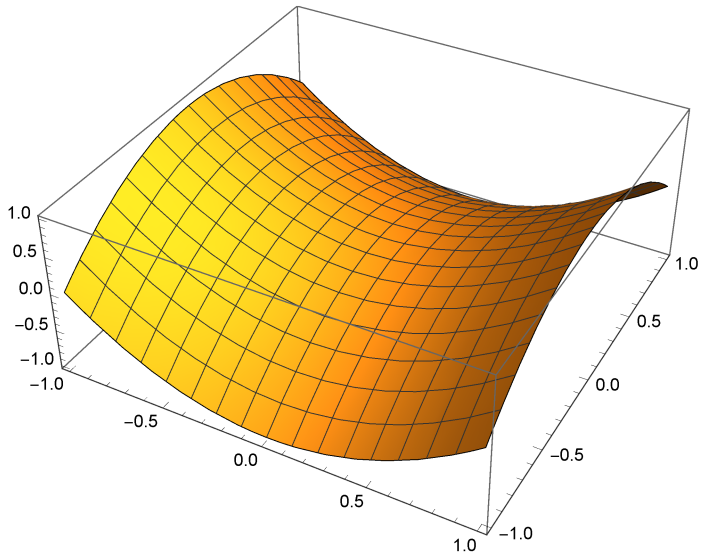
```
gradient02 = D[d0, {{x, y}}]
Solve[gradient02 == 0, {x, y}, Reals]
```

$$\left\{\frac{x}{\sqrt{x^2 + y^2}}, \frac{y}{\sqrt{x^2 + y^2}}\right\}$$

```
{}
```

```
z = x^2 - y^2
Plot3D[z, {x, -1, 1}, {y, -1, 1}]
```

$x^2 - y^2$




```
grad = D[z, {{x, y}}]
hess = D[grad, {{x, y}}];
hess // MatrixForm
```

$\{2\,x, -2\,y\}$

$$\begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix}$$

```
L = x y - λ (x + y - 4)
D[L, {{x, y, λ}}]
Solve[% == 0, {x, y, λ}]
```

$x\,y - (-4 + x + y)\,\lambda$

$\{y - \lambda, x - \lambda, 4 - x - y\}$

$\{\{x \to 2, y \to 2, \lambda \to 2\}\}$

Curvature

```
ClearAll[z, x, y, f]
D[f[x, y], {{x, y}}]
D[%, {{x, y}}] // MatrixForm
```

$$\left\{f^{(1,0)}[x, y], f^{(0,1)}[x, y]\right\}$$

$$\begin{pmatrix} f^{(2,0)}[x, y] & f^{(1,1)}[x, y] \\ f^{(1,1)}[x, y] & f^{(0,2)}[x, y] \end{pmatrix}$$

Let us return to our earlier problem.

```
z
gradient01
```

```
z
```

```
{2 x, 2 y}
```

We have found that there is a critical point at (0, 0). Now we ask how to test if it is a minimum or a maximum (or neither). Just as in the univariate case, the answer lies in the curvature of the function near the critical point. Near a maximum, a function will be concave. Near a minimum, a function will be convex. When we have access to the second derivatives of the function, we can test for concavity or convexity by looking at the matrix of second-order partial derivatives, known as the Hessian matrix. The determinant of the Hessian at a critical point is called the discriminant: if it is non-zero, then we can examine the Hessian for determinateness in order to determine curvature.

```
hessian01 = D[gradient01, {{x, y}}]
discriminant01 = Det[hessian01]
```

```
{{2, 0}, {0, 2}}
```

```
4
```

Since the determinant is positive, our critical point is a candidate maximum or minimum. (It would be negative for a saddle-point.) To determine which we have, we examine the first element of the Hessian. It is positive, so we have a minimum.

Application: Consumer Optimization

Let us solve a simple consumer optimization problem with Cobb-Douglas utility.

```
FullSimplify[ExpandAll[(a w)^a (w - a w)^(1 - a)], {w > 0, 1 > a > 0}]
```

$$-(-1+a)\left(\frac{a}{1-a}\right)^{a} w$$

```
$Assumptions = True
Clear[u, c1, c2, w, α, λ, p1, p2]
u = c1^α * c2^(1 - α)  (* utility as a function of consumption *)
e = p1 c1 + p2 c2 (* expenditure *)
bc = (w - e) (* budget constraint *)
ℒ = u + λ * bc
dℒ = D[ℒ, {{c1, c2, λ}}] // Simplify
focs = Thread[dℒ == 0] (* assume constraint binds *)
(* Mma cannot quite handle this system, but we can lend a hand *)
Eliminate[focs[[1 ;; 2]], λ]
umax`soln = Solve[% && focs[[-1]], {c1, c2}] // Flatten (* optimal consumption *)
{umax`c1, umax`c2} = {c1, c2} /. umax`soln
v = u /. umax`soln // Simplify (* indirect utility function *)
(* Roy's identity *)
{umax`c1, umax`c2} == -D[v, {{p1, p2}}] / D[v, w] // Simplify
```

```
True
```

$$c1^{\alpha}\, c2^{1-\alpha}$$

$$c1\, p1 + c2\, p2$$

$$-c1\, p1 - c2\, p2 + w$$

$$c1^{\alpha}\, c2^{1-\alpha} + (-c1\, p1 - c2\, p2 + w)\, \lambda$$

$$\left\{c1^{-1+\alpha}\, c2^{1-\alpha}\, \alpha - p1\, \lambda,\ -c1^{\alpha}\, c2^{-\alpha}\, (-1+\alpha) - p2\, \lambda,\ -c1\, p1 - c2\, p2 + w\right\}$$

$$\left\{c1^{-1+\alpha}\, c2^{1-\alpha}\, \alpha - p1\, \lambda == 0,\ -c1^{\alpha}\, c2^{-\alpha}\, (-1+\alpha) - p2\, \lambda == 0,\ -c1\, p1 - c2\, p2 + w == 0\right\}$$

$$c1^{\alpha}\, c2\, p2\, \alpha == c1^{1+\alpha}\, p1\, (1-\alpha)\ \&\&\ c1 \neq 0$$

$$\left\{c1 \to \frac{w\, \alpha}{p1},\ c2 \to -\frac{-w + w\, \alpha}{p2}\right\}$$

$$\left\{\frac{w\, \alpha}{p1},\ -\frac{-w + w\, \alpha}{p2}\right\}$$

$$\left(\frac{w\, \alpha}{p1}\right)^{\alpha} \left(\frac{w - w\, \alpha}{p2}\right)^{1-\alpha}$$

```
True
```

```
(* numerical example with plot: *)
params = {α -> 0.4, p1 → 1, p2 → 2, w → 3}
ustar = v /. params
bc
bc == 0 /. umax`soln /. params
u == ustar /. {α → 0.4}
ContourPlot[{Evaluate[bc == 0 /. params], Evaluate[u == ustar /. {α → 0.4}]},
 {c1, 0, 5}, {c2, 0, 5},
 Epilog → Point[{c1, c2} /. umax`soln /. params]]
```

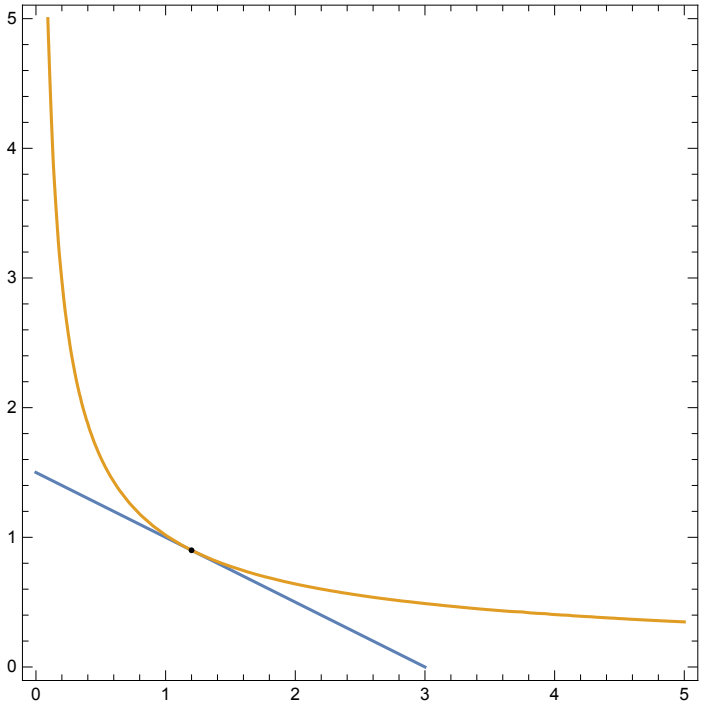
```
{α → 0.4, p1 → 1, p2 → 2, w → 3}
```

```
1.00976
```

```
- c1 p1 - c2 p2 + w
```

```
True
```

$$c1^{0.4}\ c2^{0.6} == 1.00976$$



The expenditure function is the solution to a dual minimization problem. This is implied by the indirect utility function, but we need to help *Mathematica* a little along the way.

```
(* need more info to isolate w *)
assumptions = p1 > 0 && p2 > 0 && w > 0 && 1 > α > 0
v = FullSimplify[ExpandAll[v], assumptions]
(* or v=Assuming[assumptions,Collect[v,w,Simplify]] *)
(* or Simplify[Factor[v],Assumptions→assumptions] *)
w /. Flatten@Solve[v == u0, w] (* expenditure function *)
Simplify[% == p1^α * p2^(1 - α) / (α^α * (1 - α)^(1 - α)) u0, assumptions]
```

```
p1 > 0 && p2 > 0 && w > 0 && 1 > α > 0
```

$$-\frac{w\,(-1+\alpha)\left(\frac{p2\,\alpha}{p1-p1\,\alpha}\right)^{\alpha}}{p2}$$

$$-\frac{p2\,u0\left(\frac{p2\,\alpha}{p1-p1\,\alpha}\right)^{-\alpha}}{-1+\alpha}$$

```
True
```

We can also approach this directly as a cost minimization problem.

```
u = c1^α c2^(1-α);
e = p1 c1 + p2 c2;
assumptions = p1 > 0 && p2 > 0 && 1 > α > 0 && u0 > 0;
emin`ℒ = e + λ2 (u0 - u);
D[emin`ℒ, {{c1, c2, λ2}}];
emin`focs = Thread[% == 0] (* assume constraint binds *)
(* Mma can Solve the system, but ... *)
emin`soln = Solve[emin`focs, {c1, c2, λ2}] // Flatten
(* simplification helps except with the solution for c1: *)
FullSimplify[%, Assumptions → assumptions]
(* However, ExpToTrig can help: *)
FullSimplify[% // ExpToTrig, Assumptions → assumptions]
emin = Simplify[e /. emin`soln, Assumptions → assumptions]
Simplify[emin == Power[p1/α, α] * Power[p2/(1 - α), 1 - α] u0, Assumptions → assumptions]
```

$$\left\{p1 - c1^{-1+\alpha}\, c2^{1-\alpha}\, \alpha\, \lambda 2 == 0,\ p2 - c1^{\alpha}\, c2^{-\alpha}\, (1-\alpha)\, \lambda 2 == 0,\ -c1^{\alpha}\, c2^{1-\alpha} + u0 == 0\right\}$$

$$\left\{c1 \to \left(e^{i\,\pi - i\,\pi\,\alpha + \alpha\,\text{Log}[p1] - \alpha\,\text{Log}[p2] + \alpha\,\text{Log}[-1+\alpha] - \alpha\,\text{Log}[\alpha]}\ p2\ u0\ \alpha\right) \Big/ (p1\ (-1+\alpha)),\right.$$
$$c2 \to e^{-i\,\pi + i\,\pi\,\alpha + \alpha\,\text{Log}[p1] - \alpha\,\text{Log}[p2] + \alpha\,\text{Log}[-1+\alpha] - \alpha\,\text{Log}[\alpha]}\ u0,$$
$$\left.\lambda 2 \to \frac{1}{-1+\alpha} e^{i\,\pi - i\,\pi\,\alpha + \alpha\,\text{Log}[p1] - \alpha\,\text{Log}[p2] + \alpha\,\text{Log}[-1+\alpha] - \alpha\,\text{Log}[\alpha]}\ p2\right\}$$

$$\left\{c1 \to -e^{-i\,\pi\,\alpha}\, u0 \left(\frac{p1\ (-1+\alpha)}{p2\ \alpha}\right)^{-1+\alpha},\ c2 \to u0\ (p2\ \alpha)^{-\alpha}\ (p1 - p1\ \alpha)^{\alpha},\ \lambda 2 \to -\frac{p2 \left(\frac{p1\left(-1+\frac{1}{\alpha}\right)}{p2}\right)^{\alpha}}{-1+\alpha}\right\}$$

$$\left\{c1 \to u0 \left(\frac{p1 - p1\ \alpha}{p2\ \alpha}\right)^{-1+\alpha},\ c2 \to u0\ (p2\ \alpha)^{-\alpha}\ (p1 - p1\ \alpha)^{\alpha},\ \lambda 2 \to -\frac{p2 \left(\frac{p1\left(-1+\frac{1}{\alpha}\right)}{p2}\right)^{\alpha}}{-1+\alpha}\right\}$$

$$-\frac{p2\ u0 \left(\frac{p1\left(-1+\frac{1}{\alpha}\right)}{p2}\right)^{\alpha}}{-1+\alpha}$$

```
True
```

Consumer Surplus

The compensating variation for a price change is the change in ependiture required to restore the original utility level: cv(p,p',u)=e(p',u)-e(p,u). This is the change in income required to shift the new budget line back to the old indifference curve.

```
pm01 = {α → 0.4, p1 → 1, p2 → 1, w → 100}
pm02 = {α → 0.4, p1 → 5, p2 → 1, w → 100}
u01 = v /. pm01
umax`soln
e02 = emin /. pm02 /. {u0 → u01}
e02 - w /. pm01
```

```
{α → 0.4, p1 → 1, p2 → 1, w → 100}
```

```
{α → 0.4, p1 → 5, p2 → 1, w → 100}
```

```
51.017
```

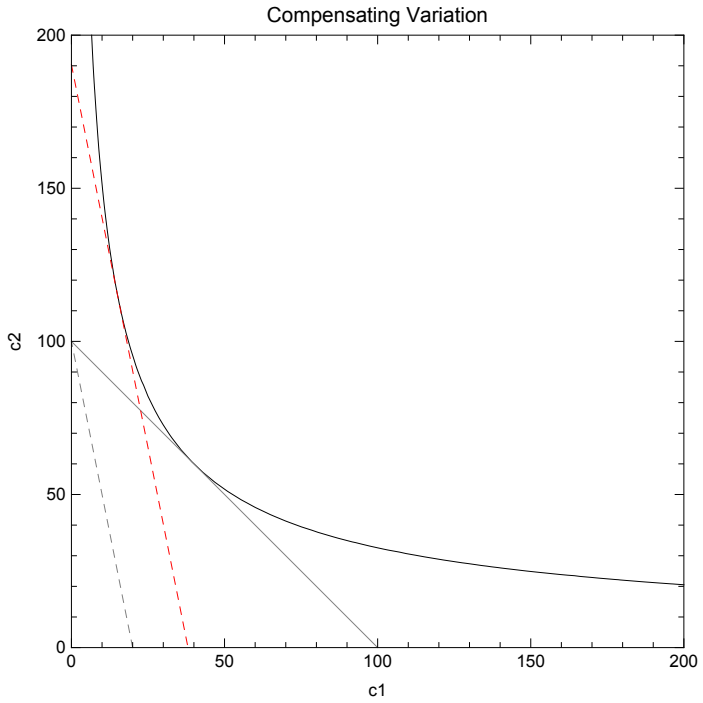
$$\left\{c1 \to \frac{w\,\alpha}{p1},\ c2 \to -\frac{-w + w\,\alpha}{p2}\right\}$$

```
190.365
```

```
90.3654
```

```
u01 == u /. pm01
ContourPlot[Evaluate[{
    u01 == u /. pm01,
    e == w /. pm01,
    e == w /. pm02,
    e == e02 /. pm02
  }],
 {c1, 0, 200}, {c2, 0, 200},
 PlotLabel → "Compensating Variation",
 FrameLabel → {"c1", "c2"},
 ContourStyle →
  {{Thin, Black}, {Thin, Gray}, {Thin, Dashed, Gray}, {Dashed, Thin, Red}},
 PlotRangePadding → None
]
```

$$51.017 == c1^{0.4}\, c2^{0.6}$$



Constrained Optimization

Lagrange Multiplier Technique

We now approach optimization subject to an equality constraint using the method of Lagrange multipliers. We focus on the case of maximizing a function of two variables, $f(x, y)$, subject to the equality constraint $g(x, y) = k$. We set up the Lagrangean as

$$\mathcal{L}(x_1, x_2, \lambda) = f(x_1, x_2) + \lambda (k - g(x_1, x_2))$$

We search for an extremum by trying to find (x_1, x_2, λ) such that the three first-order partial derivatives of the Lagrangean equal zero.

Utility Maximization

For example, suppose we want to maximize $U = x_1^{\alpha} * x_2^{1-\alpha}$ subject to an income constraint. This problem is fundamentally simple to do by hand, but it contains enough nonlinearity to create difficulties for *Mathematica*. We anticipate that by transforming the objective function. Since we know that u will be positive, we can maximize its logarithm (a strictly increasing transformation).

```
Clear[ℒ, x1, x2, α, λ, M, p1, p2]
u = 2 * x1^α * x2^(1-α)
bc = p1 * x1 + p2 * x2 - M
ℒ = Log[u] - λ * bc;
grad = D[ℒ, {{x1, x2, λ}}]
```

$$2\, x1^{\alpha}\, x2^{1-\alpha}$$

$$-M + p1\, x1 + p2\, x2$$

$$\left\{\frac{\alpha}{x1} - p1\, \lambda,\ \frac{1-\alpha}{x2} - p2\, \lambda,\ M - p1\, x1 - p2\, x2\right\}$$

In order to find potential optima, we set the gradient to zero and solve.

```
soln = Solve[grad == 0, {x1, x2, λ}, Reals][[1]]
```

$$\left\{x1 \to \frac{M\,\alpha}{p1},\ x2 \to -\frac{M\,(-1+\alpha)}{p2},\ \lambda \to \frac{1}{M}\right\}$$

```
hess = D[grad, {{x1, x2, λ}}]
```

$$\left\{\left\{-\frac{\alpha}{x1^2},\ 0,\ -p1\right\},\ \left\{0,\ -\frac{1-\alpha}{x2^2},\ -p2\right\},\ \{-p1,\ -p2,\ 0\}\right\}$$

Example

For example, if $\alpha = 0.3$, $p_1 = 1$, $p_2 = 2$, and $M = 20$, we have

```
params = {α → 0.3, p1 → 1, p2 → 2, M → 20}
"unconstrained solution:"
{u`x1, u`x2, u`λ} = {x1, x2, λ} /. soln /. params
u`x = {u`x1, u`x2}
```

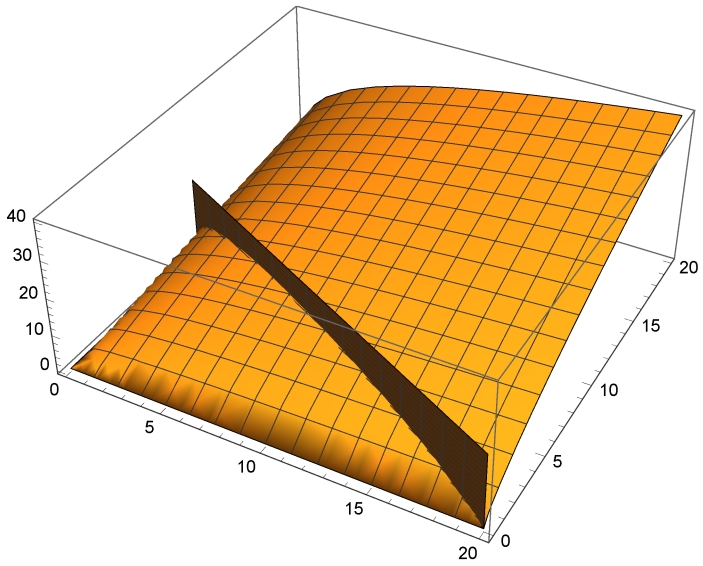
$$\{\alpha \to 0.3,\ p1 \to 1,\ p2 \to 2,\ M \to 20\}$$

unconstrained solution:

$$\left\{6.,\ 7.,\ \frac{1}{20}\right\}$$

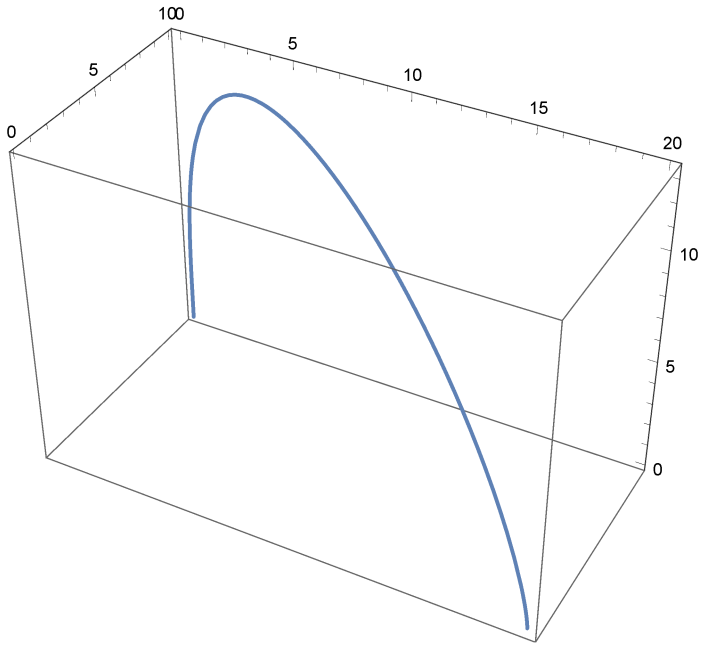
$$\{6.,\ 7.\}$$

```
cp1 = Plot3D[u /. params, {x1, 0, 20}, {x2, 0, 20}];
cp2 = ContourPlot3D[Evaluate[0 == bc /. params], {x1, 0, 20}, {x2, 0, 20}, {z, 0, 20}];
Show[{cp1, cp2}]
```



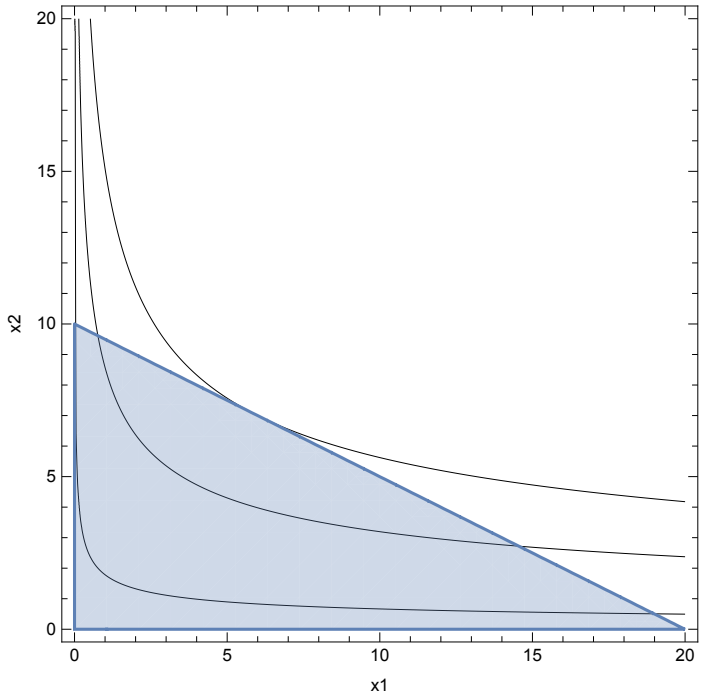

```
x2soln = Solve[0 == bc /. params, x2][[1]]
ParametricPlot3D[{x1, 10 - x1 / 2, u /. params /. {x2 → 10 - x1 / 2}}, {x1, 0, 20}]
```

$$\left\{x2 \to \frac{20 - x1}{2}\right\}$$



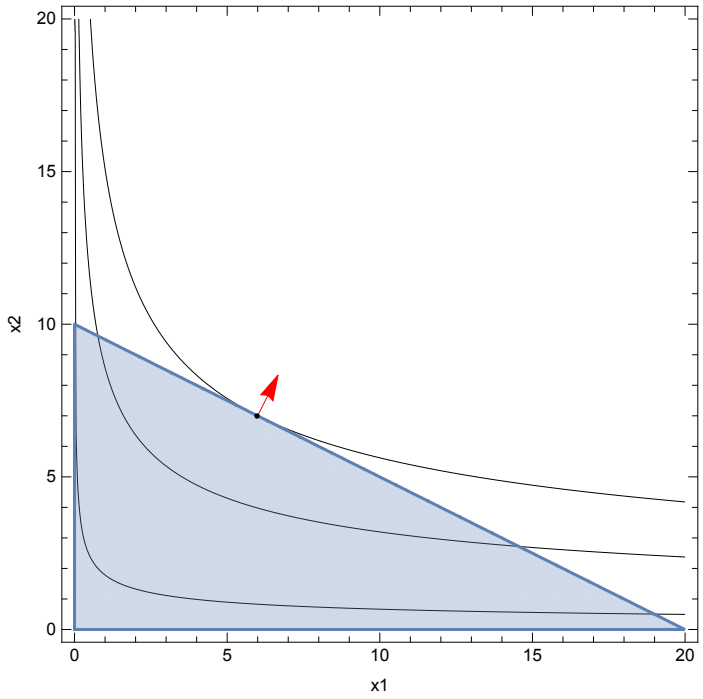
```
umax = u /. soln /. params
cp1 = ContourPlot[u /. params, {x1, 0, 20},
    {x2, 0, 20}, ContourShading → None, Contours → {3, umax, 9},
    PlotPoints → 100, ContourLabels → None, FrameLabel → {"x1", "x2"}];
cp2 = RegionPlot[Evaluate[(bc /. params) ≤ 0], {x1, 0, 20}, {x2, 0, 20}];
Show[{cp1, cp2}]
u`gradf = D[u, {{x1, x2}}] /. params /. {x1 → u`x1, x2 → u`x2}
u`gradg = D[c1, {{x1, x2}}] /. params
gu`grads =
  Graphics[{{Red, Arrow[{u`x, u`x + u`gradf}]}, Arrow[{u`x, u`x + u`gradg}]}];
Show[{cp1, cp2, gu`grads}]
```

```
13.3673
```




```
{0.668365, 1.33673}
```

```
{0, 0}
```



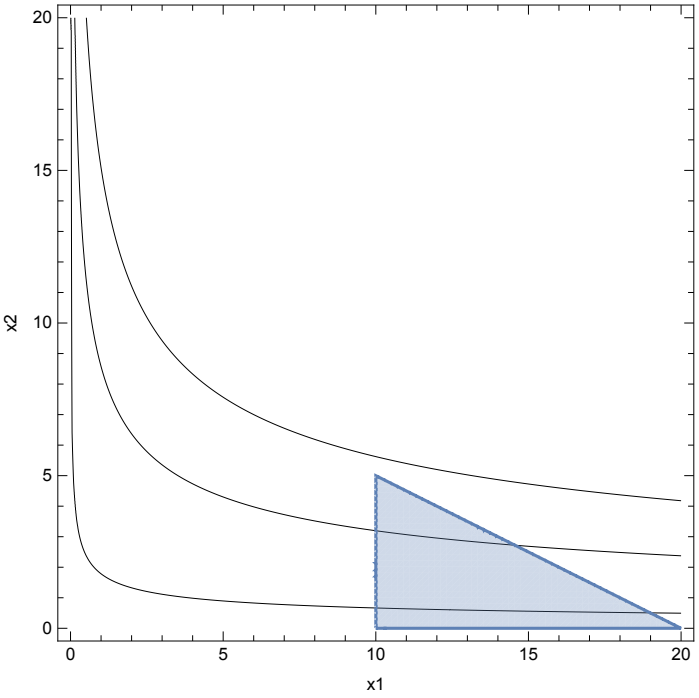
A Second Constraint: Essential Good

Suppose you need at least 10 of x1.

```
c1 = bc /. params
c2 = 10 - x1
cp3 = RegionPlot[{c1 ≤ 0 && c2 ≤ 0}, {x1, 0, 20}, {x2, 0, 20}, PlotPoints → 100];
Show[{cp1, cp3}]
```

```
-20 + x1 + 2 x2
```

```
10 - x1
```



To solve this, we need to use the Kuhn-Tucker conditions.

Motivation, we would like to make *u* as big as possible, but we impose penalties for constraint violation. If we had nonnegative penalty weights for constraint violation, we might write

```
max u(x) - λ1 g1(x) - λ2 g2(x)
```

```
objx = Log[u] - λ1 c1 - λ2 c2 /. params
dobjx = D[objx, {{x1, x2, λ1, λ2}}]
(* Solve[dobjx.{1,1,λ1,λ2}==0,{x1,x2,λ1,λ2}] *)
```

$$-(-20 + x1 + 2\,x2)\,\lambda 1 - (10 - x1)\,\lambda 2 + \text{Log}\left[2\,x1^{0.3}\,x2^{0.7}\right]$$

$$\left\{\frac{0.3}{x1^{1.}} - \lambda 1 + \lambda 2,\ \frac{0.7}{x2^{1.}} - 2\,\lambda 1,\ 20 - x1 - 2\,x2,\ -10 + x1\right\}$$

We can see that $x_1 = 10$ and $x_2 = 5$, which allows us to solve for the multipliers.

```
soln34 = Solve[dobjx[[3 ;; 4]] == 0, {x1, x2}]
Solve[dobjx[[1 ;; 2]] == 0 /. soln34[[1]], {λ1, λ2}]
```

```
{{x1 → 10, x2 → 5}}
```

```
{{λ1 → 0.07, λ2 → 0.04}}
```

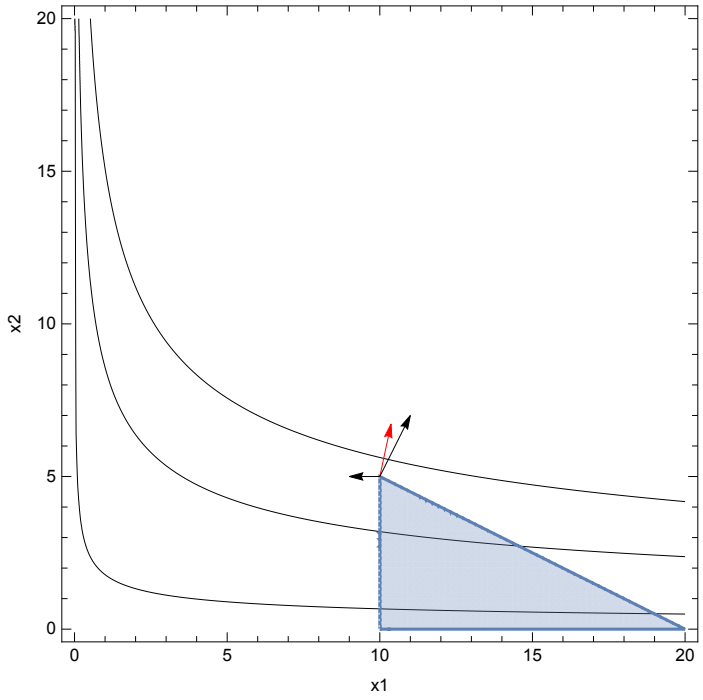
```
xsoln = {x1soln, x2soln} = {x1, x2} /. soln34[[1]]
gradf = D[u, {{x1, x2}}] /. params /. {x1 → x1soln, x2 → x2soln}
gradg1 = D[c1, {{x1, x2}}] /. {x1 → x1soln, x2 → x2soln}
gradg2 = D[c2, {{x1, x2}}] /. {x1 → x1soln, x2 → x2soln}
ggrads = Graphics[{Arrowheads[Small], {Red, Arrow[{xsoln, xsoln + gradf}]},
    Arrow[{xsoln, xsoln + gradg1}], Arrow[{xsoln, xsoln + gradg2}]}];
Show[{cp1, cp3, ggrads}]
```

```
{10, 5}
```

```
{0.369343, 1.7236}
```

```
{1, 2}
```

```
{-1, 0}
```



Cost Minimization

For a good discussion of the elasticity of substitution, see Ted Bergstrom's class notes: http://www.econ.ucsb.edu/~tedb/Courses/GraduateTheoryUCSB/elasticity%20of%20substitutionrevised.tex.pdf

We are going to work with the CES production function

```
Clear[q, n, k, a, α, ρ, w, r, λ]
assumeCES = a > 0 && n > 0 && k > 0 && 1 > α > 0 && ρ < 1
q = a * (α * k^ρ + (1 - α) n^ρ)^(1/ρ);
```

$$a > 0 \&\& n > 0 \&\& k > 0 \&\& 1 > \alpha > 0 \&\& \rho < 1$$

The CES production function represents perfect substitutes when $\rho \to 1$ and becomes a Cobb-Douglas when $\rho \to 0$. The limit as $\rho \to -\infty$ is a Leontief production function. Currently (version 10), *Mathematica* is not able to determine that.

```
Map[Limit[q, #] &, {ρ → 1, ρ → 0, ρ → -Infinity}]
```

$$\left\{a\,(n + k\,\alpha - n\,\alpha),\ a\,k^{\alpha}\,n^{1-\alpha},\ \text{Limit}\left[a\,(n^{\rho}\,(1-\alpha) + k^{\rho}\,\alpha)^{\frac{1}{\rho}},\ \rho \to -\infty\right]\right\}$$

Let us take up the case of $\rho \to 0$ in a bit more detail. It is convenient to work with the log of q.

```
lq = Log[q] // PowerExpand
```

$$\text{Log}[a] + \frac{\text{Log}[n^{\rho}\,(1-\alpha) + k^{\rho}\,\alpha]}{\rho}$$

In the resulting fraction,t is easy to see that as $\rho \to 0$ both the numerator and denominator approach 0. We therefore apply L'Hopitals rule and consider the limiting ratio of the derivatives of the numerator and denominator. This is easily seen to yield the log of the Cobb-Douglas functional form.

```
D[Log[n^ρ (1 - α) + k^ρ α], ρ]
Limit[%, ρ → 0]
```

$$\frac{k^{\rho}\,\alpha\,\text{Log}[k] + n^{\rho}\,(1-\alpha)\,\text{Log}[n]}{n^{\rho}\,(1-\alpha) + k^{\rho}\,\alpha}$$

$$\alpha\,\text{Log}[k] + \text{Log}[n] - \alpha\,\text{Log}[n]$$

Like the Cobb-Douglas, the CES production function is linear homogeneous. For example, if we double the inputs, we double the output. We have to coax *Mathematica* a bit to show this.

```
q2 = q /. {n → λ n, k → λ k}
q2 = q2 // PowerExpand
q2 = q2 // Simplify
q2 = q2 // PowerExpand
λ q == q2 // Simplify
```

$$a\,(\alpha\,(k\,\lambda)^{\rho} + (1-\alpha)\,(n\,\lambda)^{\rho})^{\frac{1}{\rho}}$$

$$a\,(n^{\rho}\,(1-\alpha)\,\lambda^{\rho} + k^{\rho}\,\alpha\,\lambda^{\rho})^{\frac{1}{\rho}}$$

$$a\,((-n^{\rho}\,(-1+\alpha) + k^{\rho}\,\alpha)\,\lambda^{\rho})^{\frac{1}{\rho}}$$

$$a\,(-n^{\rho}\,(-1+\alpha) + k^{\rho}\,\alpha)^{\frac{1}{\rho}}\,\lambda$$

True

Examine the marginal products. By inspection, they are both positive. Furthermore, their ratio depends

only on the input ratio k/n.

```
{qn, qk} = D[q, {{n, k}}] // Simplify
Simplify[qk / qn]
```

$$\left\{a\, n^{-1+\rho}\, (1-\alpha)\, \left(-n^{\rho}\, (-1+\alpha)+k^{\rho}\, \alpha\right)^{-1+\frac{1}{\rho}},\ a\, k^{-1+\rho}\, \alpha\, \left(-n^{\rho}\, (-1+\alpha)+k^{\rho}\, \alpha\right)^{-1+\frac{1}{\rho}}\right\}$$

$$\frac{k^{-1+\rho}\, n^{1-\rho}\, \alpha}{1-\alpha}$$

Since the CES function is linear homogeneous, it must satisfy Euler's theorem.

```
Simplify[q == k * qk + n * qn]
```

```
True
```

Since $\rho < 1$, the marginal products are decreasing.

```
qnn = D[qn, n] // Simplify
qkk = D[qk, k] // Simplify
```

$$-a\, k^{\rho}\, n^{-2+\rho}\, (-1+\alpha)\, \alpha\, \left(-n^{\rho}\, (-1+\alpha)+k^{\rho}\, \alpha\right)^{-2+\frac{1}{\rho}}\, (-1+\rho)$$

$$-a\, k^{-2+\rho}\, n^{\rho}\, (-1+\alpha)\, \alpha\, \left(-n^{\rho}\, (-1+\alpha)+k^{\rho}\, \alpha\right)^{-2+\frac{1}{\rho}}\, (-1+\rho)$$

We want to minimize cost, $w\,n + r\,k$, subject to achieving the given level of production q_0. We can reduce the nonlinearity of our problem by using an equivalent constraint.

```
ℒ = w * n + r * k + λ (q0^ρ - a^ρ * (α * k^ρ + (1 - α) n^ρ));
grad = D[ℒ, {{n, k, λ}}]
```

$$\left\{w-a^{\rho}\, n^{-1+\rho}\, (1-\alpha)\, \lambda\, \rho,\ r-a^{\rho}\, k^{-1+\rho}\, \alpha\, \lambda\, \rho,\ q0^{\rho}-a^{\rho}\, \left(n^{\rho}\, (1-\alpha)+k^{\rho}\, \alpha\right)\right\}$$

Unfortunately, *Mathematica* cannot quite get us there.

```
focs = Thread[grad == 0] (* constraint binds *)
Assuming[w1 > 0 && w2 > 0 && q0 > 0 && assumeCES, Solve[grad == 0, {n, k, λ}]]
```

$$\left\{w-a^{\rho}\, n^{-1+\rho}\, (1-\alpha)\, \lambda\, \rho == 0,\ r-a^{\rho}\, k^{-1+\rho}\, \alpha\, \lambda\, \rho == 0,\ q0^{\rho}-a^{\rho}\, \left(n^{\rho}\, (1-\alpha)+k^{\rho}\, \alpha\right) == 0\right\}$$

Solve::ifun :
Inverse functions are being used by Solve, so some solutions may not be found; use Reduce for complete solution information. >>

Solve::svars : Equations may not give solutions for all "solve" variables. >>

$$\left\{\left\{k \to \left(-\frac{n^{1-\rho}\, w\, \alpha}{r\, (-1+\alpha)}\right)^{\frac{1}{1-\rho}},\ \lambda \to -\frac{a^{-\rho}\, n^{1-\rho}\, w}{(-1+\alpha)\, \rho}\right\}\right\}$$

So, let us use the first two first-order conditions to eliminate λ. The result is

```
w / r == Power[n / k, ρ - 1] * (1 - α) / α
```

```
Power[α w / ((1 - α) r), 1 / (ρ - 1)] == n / k
```

$$\frac{w}{r} == \frac{\left(\frac{n}{k}\right)^{-1+\rho} (1-\alpha)}{\alpha}$$

$$\left(\frac{w\,\alpha}{r\,(1-\alpha)}\right)^{\frac{1}{-1+\rho}} == \frac{n}{k}$$

Noting that n^ρ is proportional to k^ρ, we plug into the constraint to find

```
q0^ρ == a^ρ (const * k^ρ (1 - α) + k^ρ α)
```

$$q0^\rho == a^\rho\,(\text{const}\,k^\rho\,(1-\alpha) + k^\rho\,\alpha)$$

or equivalentlly

```
ksoln01 = k → (q0 / a Power[α + (1 - α) * const, -1 / ρ] /. {const -> Power[α w / ((1 - α) r), ρ / (ρ - 1)]})
```

$$k \to \frac{q0\,\left(\alpha + (1-\alpha)\,\left(\frac{w\,\alpha}{r\,(1-\alpha)}\right)^{\frac{\rho}{-1+\rho}}\right)^{-1/\rho}}{a}$$

We can better see the link to the CES if we rewrite this as

```
ksoln02 = k -> q0 / a Power[r / α, 1 / (ρ - 1)] (α Power[r / α, ρ / (ρ - 1)] + (1 - α) Power[w / (1 - α), ρ / (ρ - 1)])^(-1/ρ)
```

$$k \to \frac{1}{a} q0\,\left(\frac{r}{\alpha}\right)^{\frac{1}{-1+\rho}} \left(\left(\frac{w}{1-\alpha}\right)^{\frac{\rho}{-1+\rho}} (1-\alpha) + \left(\frac{r}{\alpha}\right)^{\frac{\rho}{-1+\rho}} \alpha\right)^{-1/\rho}$$

By symmetry we expect

```
wsoln02 = n -> q0 / a Power[w / (1 - α), 1 / (ρ - 1)] (α Power[r / α, ρ / (ρ - 1)] + (1 - α) Power[w / (1 - α), ρ / (ρ - 1)])^(-1/ρ)
```

$$n \to \frac{1}{a} q0\,\left(\frac{w}{1-\alpha}\right)^{\frac{1}{-1+\rho}} \left(\left(\frac{w}{1-\alpha}\right)^{\frac{\rho}{-1+\rho}} (1-\alpha) + \left(\frac{r}{\alpha}\right)^{\frac{\rho}{-1+\rho}} \alpha\right)^{-1/\rho}$$

Substitute into $r\,k + w\,n$.

```
r k + w n /. ksoln02 /. wsoln02 // Simplify
```

$$\frac{1}{a} q0\,\left(w\,\left(\frac{w}{1-\alpha}\right)^{\frac{1}{-1+\rho}} + r\,\left(\frac{r}{\alpha}\right)^{\frac{1}{-1+\rho}}\right) \left(w\,\left(\frac{w}{1-\alpha}\right)^{\frac{1}{-1+\rho}} + \left(\frac{r}{\alpha}\right)^{\frac{\rho}{-1+\rho}} \alpha\right)^{-1/\rho}$$

To better see the link with the CES function, note that

```
(w (w / (1 - α))^(1 / (-1 + ρ)) + r (r / α)^(1 / (-1 + ρ))) (w (w / (1 - α))^(1 / (-1 + ρ)) + (r / α)^(ρ / (-1 + ρ)) α)^(-1/ρ) ==
    Power[(1 - α) (w / (1 - α))^(ρ / (ρ - 1)) + α (r / α)^(ρ / (ρ - 1)), (ρ - 1) / ρ] // PowerExpand // Simplify
```

```
True
```

Note that since cost is proportional to output, we have constant marginal cost.

Additional Topics

Relations

Cartesian Product

Cartesian Product using Tuples

Mathematica does not offer a specialized Cartesian product command, but the functionality exists in the Tuples command. Again, be aware that Tuples does not delete duplicates.

```
Tuples[{{0, 0, 1}, {2, 3}}]
Tuples[DeleteDuplicates /@ {{0, 0, 1}, {2, 3}}]
```

```
{{0, 2}, {0, 3}, {0, 2}, {0, 3}, {1, 2}, {1, 3}}
```

```
{{0, 2}, {0, 3}, {1, 2}, {1, 3}}
```

Cartesian Product using Outer and Flatten

The Outer command takes as inputs a function and lists of arguments. It applies the function to all possible argument combinations, taking the first argument from the first list, the second from the second list, and so forth.

```
Clear[f, a, b, c, d]
Outer[f, {a, b}, {c, d}]
```

```
{{f[a, c], f[a, d]}, {f[b, c], f[b, d]}}
```

Note how the result is broken up into sublists by the value of the first argument. If we do not want that, we can use Flatten to change the structure to a simple list.

```
Clear[f, a, b, c, d]
Outer[f, {a, b}, {c, d}] // Flatten
```

```
{f[a, c], f[a, d], f[b, c], f[b, d]}
```

We can Flatten lists to various levels:

```
Flatten[{0, {1}, {3, {4}}, {5, {6}, {7, {8}}}}, 1]
Flatten[{0, {1}, {3, {4}}, {5, {6}, {7, {8}}}}, 2]
Flatten[{0, {1}, {3, {4}}, {5, {6}, {7, {8}}}}, 3]
```

```
{0, 1, 3, {4}, 5, {6}, {7, {8}}}
```

```
{0, 1, 3, 4, 5, 6, 7, {8}}
```

```
{0, 1, 3, 4, 5, 6, 7, 8}
```

By choosing the List command as our function, the Outer command can also be used to produce a Cartesian product. But since it will return a list of lists of lists, you will probably want to Flatten the first level.

```
cp = Outer[List, {0, 1}, {2, 3}]
Flatten[cp, 1]
```

```
{{{0, 2}, {0, 3}}, {{1, 2}, {1, 3}}}
```

```
{{0, 2}, {0, 3}, {1, 2}, {1, 3}}
```

Relations

Random Relation

Start with a set direct product (Cartesian product):

```
n6 = Range[6];
n6sq = Tuples[{n6, n6}];
Grid[Partition[n6sq, 6]]
```

```
{1, 1}  {1, 2}  {1, 3}  {1, 4}  {1, 5}  {1, 6}
{2, 1}  {2, 2}  {2, 3}  {2, 4}  {2, 5}  {2, 6}
{3, 1}  {3, 2}  {3, 3}  {3, 4}  {3, 5}  {3, 6}
{4, 1}  {4, 2}  {4, 3}  {4, 4}  {4, 5}  {4, 6}
{5, 1}  {5, 2}  {5, 3}  {5, 4}  {5, 5}  {5, 6}
{6, 1}  {6, 2}  {6, 3}  {6, 4}  {6, 5}  {6, 6}
```

Any subset of the set direct product is a binary relation.

```
randomrelation = RandomSample[n6sq, 18]
subsetQ[n6sq, randomrelation]
```

```
{{4, 1}, {6, 2}, {5, 4}, {6, 1}, {5, 1}, {2, 2}, {1, 1}, {3, 1}, {4, 4},
 {1, 5}, {2, 5}, {3, 5}, {3, 3}, {1, 2}, {2, 3}, {2, 1}, {4, 6}, {1, 4}}
```

```
subsetQ[{{1, 1}, {1, 2}, {1, 3}, {1, 4}, {1, 5}, {1, 6}, {2, 1}, {2, 2},
  {2, 3}, {2, 4}, {2, 5}, {2, 6}, {3, 1}, {3, 2}, {3, 3}, {3, 4}, {3, 5},
  {3, 6}, {4, 1}, {4, 2}, {4, 3}, {4, 4}, {4, 5}, {4, 6}, {5, 1}, {5, 2}, {5, 3},
  {5, 4}, {5, 5}, {5, 6}, {6, 1}, {6, 2}, {6, 3}, {6, 4}, {6, 5}, {6, 6}},
 {{4, 1}, {6, 2}, {5, 4}, {6, 1}, {5, 1}, {2, 2}, {1, 1}, {3, 1}, {4, 4},
  {1, 5}, {2, 5}, {3, 5}, {3, 3}, {1, 2}, {2, 3}, {2, 1}, {4, 6}, {1, 4}}]
```

Adjacency Matrix

Adjacency Matrix Representation

We can represent any finite binary relation with an adjacency matrix. This gives a boolean matrix representation of the binary relation (X, Y, R), where matrix element a_{ij} is 1 if $(x_i, y_j) \in R$ and is otherwise 0.

```
boolrep = If[MemberQ[randomrelation, #], 1, 0] & /@ n6sq;
adjacencyMatrix = Partition[boolrep, 6];
MatrixForm[adjacencyMatrix]
```

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

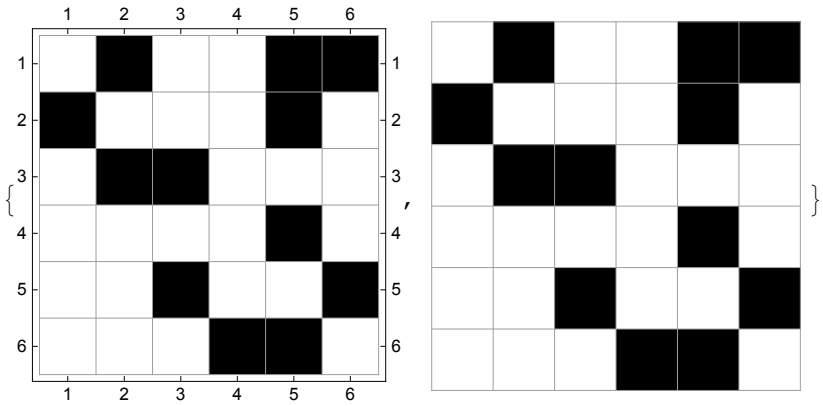
Visual Representations

Matrix Plot

An adjacency matrix can be given a nice visual representation using the MatrixPlot command. (The ArrayPlot command could also be used.)

```
(* matrix plot of a random relation *)
adjacencyMatrix = RandomChoice[{0.6, 0.4} → {0, 1}, {6, 6}]
{
 MatrixPlot[adjacencyMatrix, Mesh → True,
   ImageSize → 200, ColorFunction → "Monochrome"],
 ArrayPlot[adjacencyMatrix, Mesh → True,
   ImageSize → 200, ColorFunction → "Monochrome"]
}
```

```
{{0, 1, 0, 0, 1, 1}, {1, 0, 0, 0, 1, 0}, {0, 1, 1, 0, 0, 0},
 {0, 0, 0, 0, 1, 0}, {0, 0, 1, 0, 0, 1}, {0, 0, 0, 1, 1, 0}}
```

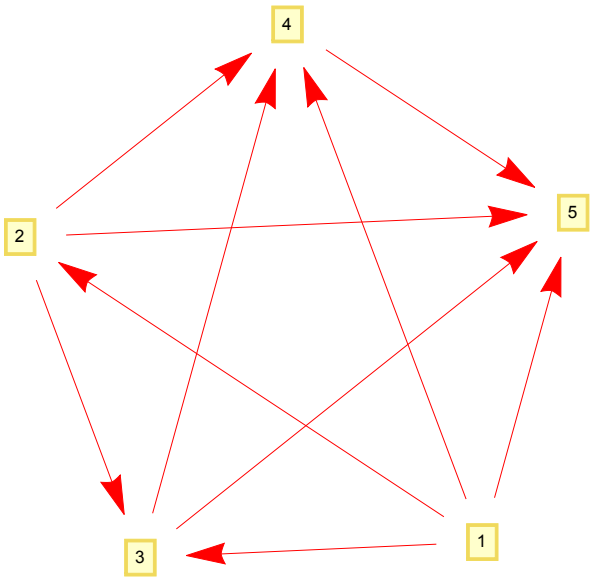


Directed Graph Display

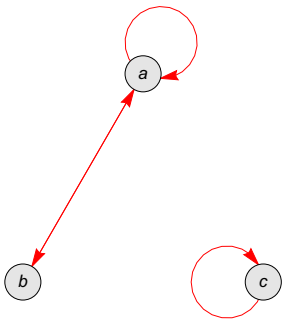
A binary relation on a set *X* can be nicely represented as a directed graph.

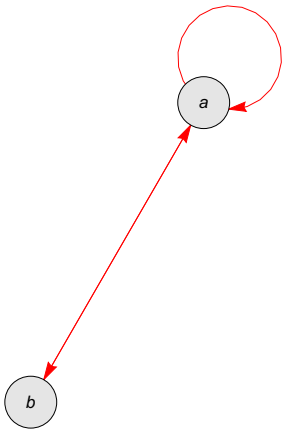
```
adjacencyMatrix = Table[If[i ≤ j, 1, 0], {i, 5}, {j, 5}];
% // MatrixForm
GraphPlot[adjacencyMatrix,
 DirectedEdges → True,
 EdgeRenderingFunction → ({Red, Arrowheads[Large], Arrow[#1, 0.1]} &),
 VertexLabeling → True]
```

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$



```
abcGraphPlot[rules_] := GraphPlot[rules,
  VertexLabeling → True,
  MultiedgeStyle → False,
  EdgeRenderingFunction → ({Red, Arrowheads[0.05], Arrow[#1, 0.15]} &),
  DirectedEdges → True,
  SelfLoopStyle → 0.3,
  VertexCoordinateRules → {a → {1, Sqrt[3]}, b → {0, 0}, c → {2, 0}},
  VertexRenderingFunction →
   ({GrayLevel[0.9], EdgeForm[Black], Disk[#, .15], Black, Text[#2, #1]} &),
  ImagePadding → 10,
  ImageSize → 200]
abcGraphPlot[{a → a, a → b, b → a, c → c}]
abcGraphPlot[{a → a, a → b, b → a}]
```





```
? Gray
```

Info-a9239a23-5ddd-4fe4-aa32-11be485c2603

Gray represents the color gray in graphics or style specifications. ≫

```
? Integer
```

Info-e9142e5f-ee1c-4239-a322-8cec97ac9586

Integer is the head used for integers. ≫

```
? Disk
```

Info-761ef30e-c1ab-40aa-bf23-9b718c36e9c1

Disk[{x, y}, r] represents a disk of radius r centered at {x, y}.

Disk[{x, y}] gives a disk of radius 1.

Disk[{x, y}, {r_x, r_y}] gives an axis–aligned elliptical disk with semiaxes lengths r_x and r_y.

Disk[{x, y}, …, {θ_1, θ_2}] gives a sector of a disk from angle θ_1 to θ_2. ≫

```
? GraphPlot
```

Info-9a5b3143-e16a-4444-9974-c867d56b4e7f

GraphPlot[g] generates a plot of the graph g.

GraphPlot[{$v_{i1} \to v_{j1}$, $v_{i2} \to v_{j2}$, …}] generates a plot of the graph in which vertex v_{ik} is connected to vertex v_{jk}.

GraphPlot[{{$v_{i1} \to v_{j1}$, lbl_1}, …}] associates labels lbl_k with edges in the graph.

GraphPlot[m] generates a plot of the graph represented by the adjacency matrix m. ≫

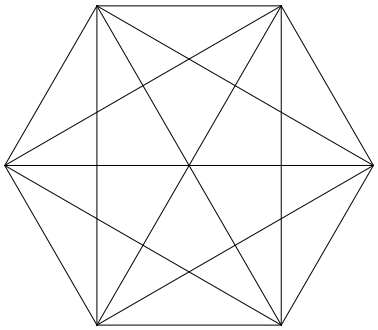
More Graphs

Mathematica supports: graph product, lexicographic product, rooted product, tensor product, graph sum, graph difference, graph power, graph join, graph intersection, and graph union.

A complete graph has an edge between each pair of vertices. We can of course construct this “by hand”:

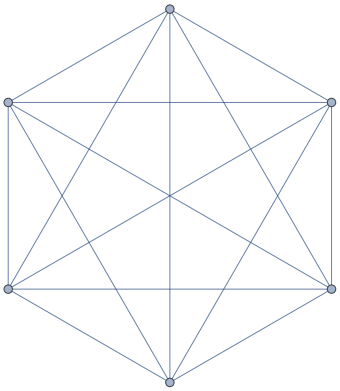
```
Clear[n, step, points, points2pairs, lines]; n = 6; step = 2 * π / n;
points = Table[{Cos[step * i], Sin[step * i]}, {i, n}]
points2pairs = Subsets[points, {2}];
lines = Line[points2pairs];
Graphics[lines, ImageSize → {200, 200}]
```

$$\left\{\left\{\frac{1}{2}, \frac{\sqrt{3}}{2}\right\}, \left\{-\frac{1}{2}, \frac{\sqrt{3}}{2}\right\}, \{-1, 0\}, \left\{-\frac{1}{2}, -\frac{\sqrt{3}}{2}\right\}, \left\{\frac{1}{2}, -\frac{\sqrt{3}}{2}\right\}, \{1, 0\}\right\}$$



But we may want to accept the Mathematica defaults:

```
Show[
 CompleteGraph[6],
 ImageSize → Small
]
```



Introduction to Polynomials

```
Solve[a * x^2 + b * x + c == 0, x]
```

$$\left\{\left\{x \to \frac{-b - \sqrt{b^2 - 4\,a\,c}}{2\,a}\right\}, \left\{x \to \frac{-b + \sqrt{b^2 - 4\,a\,c}}{2\,a}\right\}\right\}$$

Mathematica can also provide general solutions for the cubic and quartic cases. However that is as far as a general solution goes, since Galois and Abel

showed early in the 19th century that a general solution for quintics is impossible. However, special cases are solvable.

```
polyrule = {c0 → 0, c1 → 1, c2 → 2, c3 → 3, c4 → 4, c5 → 5}
Solve[c5 * x^5 + c4 * x^4 + c3 * x^3 + c2 * x^2 + c1 * x + c0 == 0 /. polyrule, x]
```

```
{c0 → 0, -20 + x1 + 2 x2 → 1, 10 - x1 → 2, c3 → 3, c4 → 4, c5 → 5}
```

$$\left\{\{x \to 0\},\right.$$

$$\{x \to \frac{1}{10}\left(-2+5\sqrt{\left(-\frac{6}{25}+\frac{1}{(-5+10\,i)^{1/3}}+\frac{(-1+2\,i)^{1/3}}{5^{2/3}}\right)}-\sqrt{\left(\frac{1}{5}\left(-60-25\,(-5+10\,i)^{1/3}-\frac{25\times 5^{2/3}}{(-1+2\,i)^{1/3}}-56\Big/\left(\sqrt{\left(-\frac{6}{25}+\frac{1}{(-5+10\,i)^{1/3}}+\frac{(-1+2\,i)^{1/3}}{5^{2/3}}\right)}\right)\right)\right)}\right)\},$$

$$\{x \to \frac{1}{10}\left(-2+5\sqrt{\left(-\frac{6}{25}+\frac{1}{(-5+10\,i)^{1/3}}+\frac{(-1+2\,i)^{1/3}}{5^{2/3}}\right)}+\sqrt{\left(\frac{1}{5}\left(-60-25\,(-5+10\,i)^{1/3}-\frac{25\times 5^{2/3}}{(-1+2\,i)^{1/3}}-56\Big/\left(\sqrt{\left(-\frac{6}{25}+\frac{1}{(-5+10\,i)^{1/3}}+\frac{(-1+2\,i)^{1/3}}{5^{2/3}}\right)}\right)\right)\right)}\right)\},$$

$$\{x \to \frac{1}{10}\left(-2-5\sqrt{\left(-\frac{6}{25}+\frac{1}{(-5+10\,i)^{1/3}}+\frac{(-1+2\,i)^{1/3}}{5^{2/3}}\right)}-\sqrt{\left(\frac{1}{5}\left(-60-25\,(-5+10\,i)^{1/3}-\frac{25\times 5^{2/3}}{(-1+2\,i)^{1/3}}+56\Big/\left(\sqrt{\left(-\frac{6}{25}+\frac{1}{(-5+10\,i)^{1/3}}+\frac{(-1+2\,i)^{1/3}}{5^{2/3}}\right)}\right)\right)\right)}\right)\},$$

$$\left.\{x \to \frac{1}{10}\left(-2-5\sqrt{\left(-\frac{6}{25}+\frac{1}{(-5+10\,i)^{1/3}}+\frac{(-1+2\,i)^{1/3}}{5^{2/3}}\right)}+\sqrt{\left(\frac{1}{5}\left(-60-25\,(-5+10\,i)^{1/3}-\frac{25\times 5^{2/3}}{(-1+2\,i)^{1/3}}+56\Big/\left(\sqrt{\left(-\frac{6}{25}+\frac{1}{(-5+10\,i)^{1/3}}+\frac{(-1+2\,i)^{1/3}}{5^{2/3}}\right)}\right)\right)\right)}\right)\}\right\}$$

In addition, we can always expect that finding an approximate numerical solution is possible.

```
NSolve[c5 * x^5 + c4 * x^4 + c3 * x^3 + c2 * x^2 + c1 * x + c0 == 0 /. polyrule, x] // TableForm
```

```
x → -0.537832 - 0.358285 i
x → -0.537832 + 0.358285 i
x → 0.
x → 0.137832 - 0.678154 i
x → 0.137832 + 0.678154 i
```

Sequences

Generating Finite Sequences

Mathematica makes it easy to generate finite sequences.

Basic use of the `Table` command generates sequences from an expression given an iterator. The iterator is a list containing the variable in the expression, the initial value of the iterator variable, and the

final value of the iterator variable.

```
Table[1/n, {n, 1, 10}]
```

$$\left\{1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}, \frac{1}{7}, \frac{1}{8}, \frac{1}{9}, \frac{1}{10}\right\}$$

Basic use of the `RecurrenceTable` command generates sequences for a recurrence relations.

```
Clear[a]
RecurrenceTable[{a[t] == 0.9 a[t - 1], a[0] == 100}, a, {t, 0, 10}]
```

```
{100., 90., 81., 72.9, 65.61, 59.049, 53.1441, 47.8297, 43.0467, 38.742, 34.8678}
```

When the recurrence relation is linear, the `LinearRecurrence` command is an alternative to the use of `RecurrenceTable`.

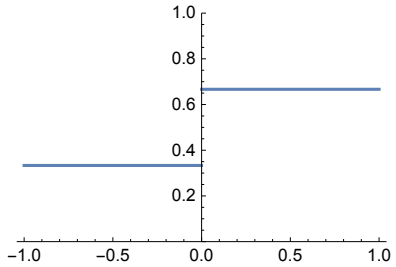
```
LinearRecurrence[{0.9}, {100}, 11]
```

```
{100., 90., 81., 72.9, 65.61, 59.049, 53.1441, 47.8297, 43.0467, 38.742, 34.8678}
```

Limits

Mathematica usally takes limits from the right. However you can instruct *Mathematica* to take the limit from the left by using the Direction option.

```
ClearAll[f, x]
f[x_] := Piecewise[{{1/3, x < 0}, {2/3, x ≥ 0}}]
Plot[f[x], {x, -1, 1}, PlotRange → {0, 1}, ImageSize → 200]
{
 Limit[f[x], {x → 0}, Direction → 1] (* limit from left *),
 Limit[f[x], {x → 0}, Direction → -1] (* limit from right *),
 Limit[f[x], {x → 0}] (* limit from right (default) *),
 Limit[f[x], {x → ∞}] (* limit from left (because only sensible) *),
 Limit[f[x], {x → -∞}] (* limit from right *)
}
```



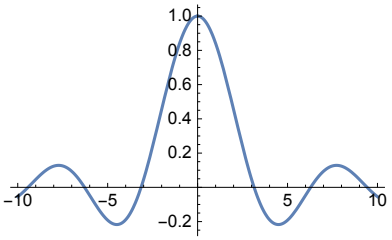
$$\left\{\left\{\frac{1}{3}\right\}, \left\{\frac{2}{3}\right\}, \left\{\frac{2}{3}\right\}, \left\{\frac{2}{3}\right\}, \left\{\frac{1}{3}\right\}\right\}$$

We say the limit exists if and only if it exists and is the same from the left and the right. So we see that $f(x)$ does not have a limit at $x = 0$.

A function need not be defined at a point in order to have a well-defined limit there. Consider two classic example of sin(*x*)/*x*. The plot suggests that although the function is undefined at *x* = 0 it still has a well-defined limit there, and we confirm this with the Limit function.

```
Plot[Sin[x]/x, {x, -10, 10}, Exclusions → {x == 0}, ImageSize → 200]
```

```
{Limit[Sin[x]/x, {x → 0}, Direction → 1], Limit[Sin[x]/x, {x → 0}]}
```

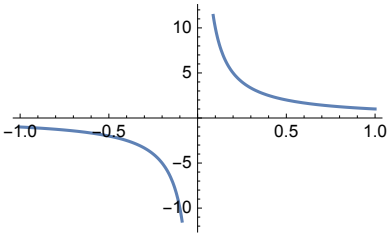



```
{{1}, {1}}
```

We see that the limit from the left matches the limit from the right for sin(*x*)/*x*. In contrast, consider 1/*x*. It too is undefined at *x* = 0, but it does not have a limit there. See why by examining the plot.

```
Plot[1/x, {x, -1, 1}, Exclusions → {x == 0}, ImageSize → 200]
```

```
{Limit[1/x, {x → 0}, Direction → 1], Limit[1/x, {x → 0}]}
```




```
{{-∞}, {∞}}
```

Parameter Dependent Limits

If an expression includes a parameter, the limiting value may naturally depend on that parameter.Torrence and Torrence (p.198) offer the following example.

```
Clear[x, n]
```

```
Limit[(1 - x^n)/n, {x → 0}]
```

```
{Limit[(1 - x^n)/n, x → 0]}
```

However we can pin down the limit if we know whether *n* is positive or negative. We can use the Assumptions option to use this information.

```
{Limit[(1 - x^n)/n, {x → 0}, Assumptions → {n < 0}],
 Limit[(1 - x^n)/n, {x → 0}, Assumptions → {n > 0}]}
```

```
{{∞}, {1/n}}
```

Logistic Map

The logistic function and its fixed points:

```
ClearAll[f, a, x]
f[x_] := a x (1 - x)
fp1 = Solve[x == f[x], x]
```

```
{{x → 0}, {x → (-1 + a)/a}}
```

Stability of the steady state: we need a slope less than unity in absolute value:

```
D[f[x], x] // Simplify
% /. fp1 // Simplify
```

```
a - 2 a x
```

```
{a, 2 - a}
```

So for $a \in (0, 1)$, the stable fixed point is 0. But for $a \in (1, 3)$, the stable fixed point is $(a - 1)/a$.

However, something interesting happens as we pass 3. We can see by composing the logistic function with itself that there are now two more fixed points of f°f.

```
f2[x_] := f[f[x]]
Collect[f2[x], x]
fp2 = Solve[x == f2[x], x]
```

```
-3 - 2 a + a^2 == (a - 3) (a + 1) // Simplify
```

```
a^2 x + (-1 - a) a^2 x^2 + 2 a^3 x^3 - a^3 x^4
```

```
{{x → 0}, {x → (-1 + a)/a}, {x → (a + a^2 - a Sqrt[-3 - 2 a + a^2])/(2 a^2)}, {x → (a + a^2 + a Sqrt[-3 - 2 a + a^2])/(2 a^2)}}
```

```
True
```

Since we already knew about 0 and $1 - 1/a$, solving directly is really too much work. Let us instead form the polynomial whose roots we're seeking and divide it by the polynomial whose roots we've found:

```
poly4fp1 = Collect[Times @@ (x - (x /. fp1)), x]
PolynomialQuotientRemainder[x - f2[x], poly4fp1, x] // Simplify
poly4fp2 = %[[1]]
Solve[poly4fp2 == 0, x]
```

$$\frac{(1-a)\,x}{a} + x^2$$

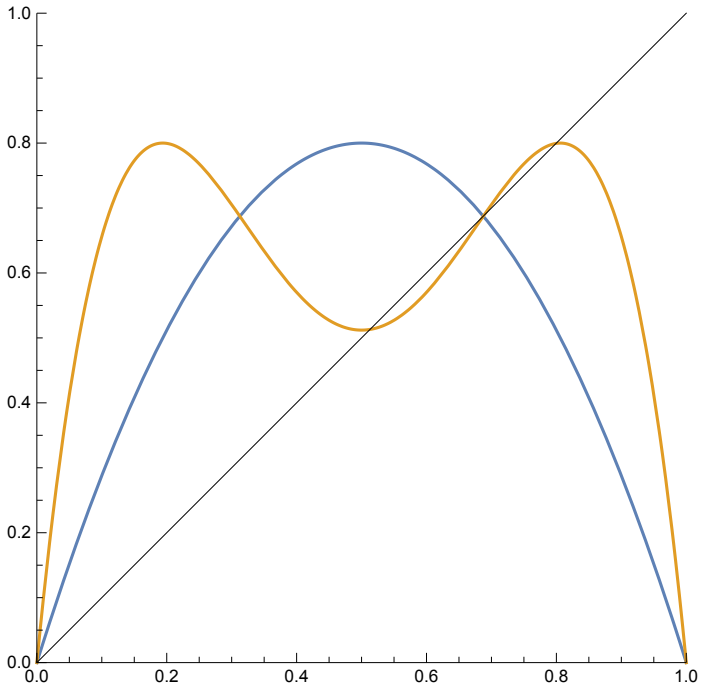
$$\left\{a\left(1+a-a\,x+a^2\,(-1+x)\,x\right),\ 0\right\}$$

$$a\left(1+a-a\,x+a^2\,(-1+x)\,x\right)$$

$$\left\{\left\{x \to \frac{a+a^2-a\sqrt{-3-2\,a+a^2}}{2\,a^2}\right\},\ \left\{x \to \frac{a+a^2+a\sqrt{-3-2\,a+a^2}}{2\,a^2}\right\}\right\}$$

Note that we do not get new fixed points of f°f until we reach $a = 3.0$. Then suddenly we have twice as many.

```
g1 = Plot[Evaluate[{f[x], f2[x]} /. {a → 3.2}], {x, 0, 1},
   PlotRange → {{0, 1}, {0, 1}},
   AspectRatio → 1,
   Epilog → Line[{{0, 0}, {1, 1}}]
 ]
Export[mathimages <> "logistic-ff2.pdf", g1];
```



StringJoin::string : String expected at position 1 in mathimages <> logistic–ff2.pdf. >>

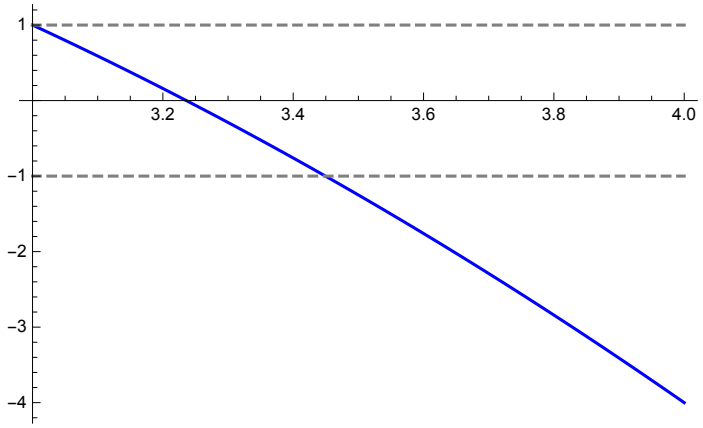
Export::chtype : First argument mathimages <> logistic–ff2.pdf is not a valid file specification. >>

What's more, although our old stable fixed point is now unstable, the two new points are initially stable.

In an interesting bit of symmetry, the slope of $f \circ f$ is the same at each of the new fixed points.

```
D[f2[x], x] /. fp2 // Simplify
Plot[{4 + 2 a - a^2, 1, -1}, {a, 3, 4},
 PlotStyle → {Blue, {Gray, Dashed}, {Gray, Dashed}}]
Solve[4 + 2 a - a^2 == -1, a]
% // N
```

$$\{a^2,\ (-2+a)^2,\ 4+2a-a^2,\ 4+2a-a^2\}$$



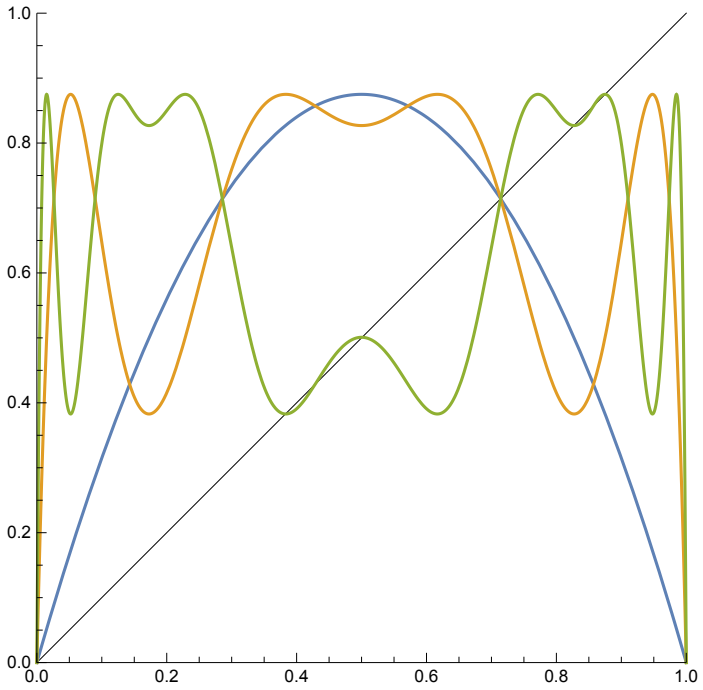
$$\{\{a \to 1-\sqrt{6}\},\ \{a \to 1+\sqrt{6}\}\}$$

```
{{a → -1.44949}, {a → 3.44949}}
```

We can again proceed in the same fasion. But it is $f^{\circ 4}$ not $f^{\circ 3}$ that provides the next periodic attractors.

```
f3[x_] := f[f[f[x]]]
f4[x_] := Nest[f, x, 4]
Collect[f3[x], x]
Plot[Evaluate[{f[x], f3[x], f4[x]} /. {a → 3.5}], {x, 0, 1},
 AspectRatio → 1, PlotRange → {{0, 1}, {0, 1}}, Prolog → Line[{{0, 0}, {1, 1}}]]
```

$$a^3 x + a^3 \left(-1 - a - a^2\right) x^2 + a^3 \left(2 a + 2 a^2 + 2 a^3\right) x^3 +$$
$$a^3 \left(-a - a^2 - 6 a^3 - a^4\right) x^4 + a^3 \left(6 a^3 + 4 a^4\right) x^5 + a^3 \left(-2 a^3 - 6 a^4\right) x^6 + 4 a^7 x^7 - a^7 x^8$$



```
PolynomialQuotientRemainder[x - f4[x], poly4fp1 * poly4fp2, x]
poly4fp4 = Collect[%[[1]], x]
Solve[poly4fp4 == 0, x] // Simplify;
% /. {{a → 3.5}}
```

$$\left\{1 + a^2 + \left(-a^2 - a^3 - a^4 - a^5\right) x + \left(2 a^3 + a^4 + 4 a^5 + a^6 + 2 a^7\right) x^2 + \left(-a^3 - 5 a^5 - 4 a^6 - 5 a^7 - 4 a^8 - a^9\right) x^3 + \left(2 a^5 + 6 a^6 + 4 a^7 + 14 a^8 + 5 a^9 + 3 a^{10}\right) x^4 + \left(-4 a^6 - a^7 - 18 a^8 - 12 a^9 - 12 a^{10} - 3 a^{11}\right) x^5 + \left(a^6 + 10 a^8 + 17 a^9 + 18 a^{10} + 15 a^{11} + a^{12}\right) x^6 + \left(-2 a^8 - 14 a^9 - 12 a^{10} - 30 a^{11} - 6 a^{12}\right) x^7 + \left(6 a^9 + 3 a^{10} + 30 a^{11} + 15 a^{12}\right) x^8 + \left(-a^9 - 15 a^{11} - 20 a^{12}\right) x^9 + \left(3 a^{11} + 15 a^{12}\right) x^{10} - 6 a^{12} x^{11} + a^{12} x^{12}, 0\right\}$$

$$1 + a^2 + \left(-a^2 - a^3 - a^4 - a^5\right) x + \left(2 a^3 + a^4 + 4 a^5 + a^6 + 2 a^7\right) x^2 + \left(-a^3 - 5 a^5 - 4 a^6 - 5 a^7 - 4 a^8 - a^9\right) x^3 + \left(2 a^5 + 6 a^6 + 4 a^7 + 14 a^8 + 5 a^9 + 3 a^{10}\right) x^4 + \left(-4 a^6 - a^7 - 18 a^8 - 12 a^9 - 12 a^{10} - 3 a^{11}\right) x^5 + \left(a^6 + 10 a^8 + 17 a^9 + 18 a^{10} + 15 a^{11} + a^{12}\right) x^6 + \left(-2 a^8 - 14 a^9 - 12 a^{10} - 30 a^{11} - 6 a^{12}\right) x^7 + \left(6 a^9 + 3 a^{10} + 30 a^{11} + 15 a^{12}\right) x^8 + \left(-a^9 - 15 a^{11} - 20 a^{12}\right) x^9 + \left(3 a^{11} + 15 a^{12}\right) x^{10} - 6 a^{12} x^{11} + a^{12} x^{12}$$

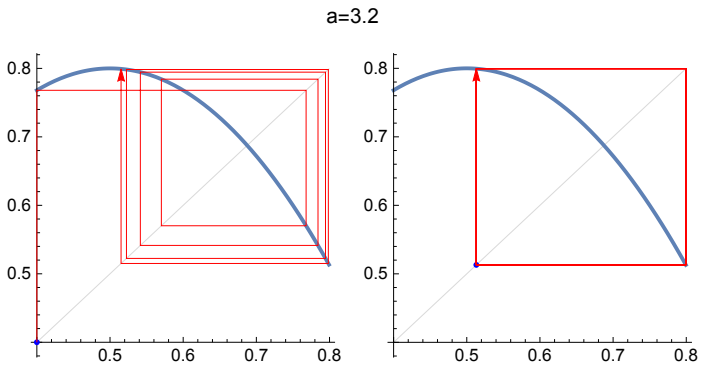
```
{{{x → 0.38282}, {x → 0.500884}, {x → 0.826941},
   {x → 0.874997}, {x → 0.049385 - 0.0241573 i}, {x → 0.049385 + 0.0241573 i},
   {x → 0.166354 - 0.0761994 i}, {x → 0.166354 + 0.0761994 i},
   {x → 0.505703 - 0.177965 i}, {x → 0.505703 + 0.177965 i},
   {x → 0.985737 - 0.00710474 i}, {x → 0.985737 + 0.00710474 i}}}
```

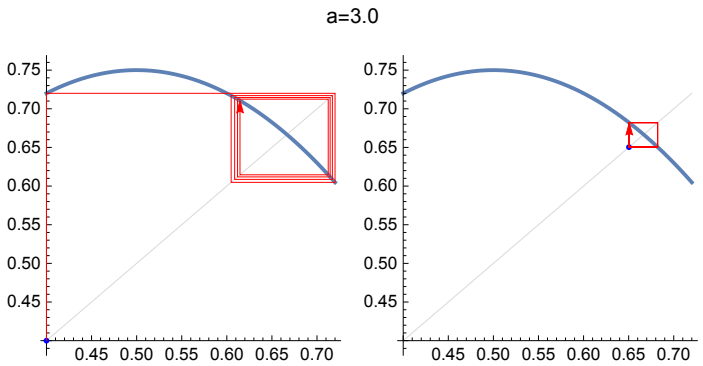
Cobweb Plots

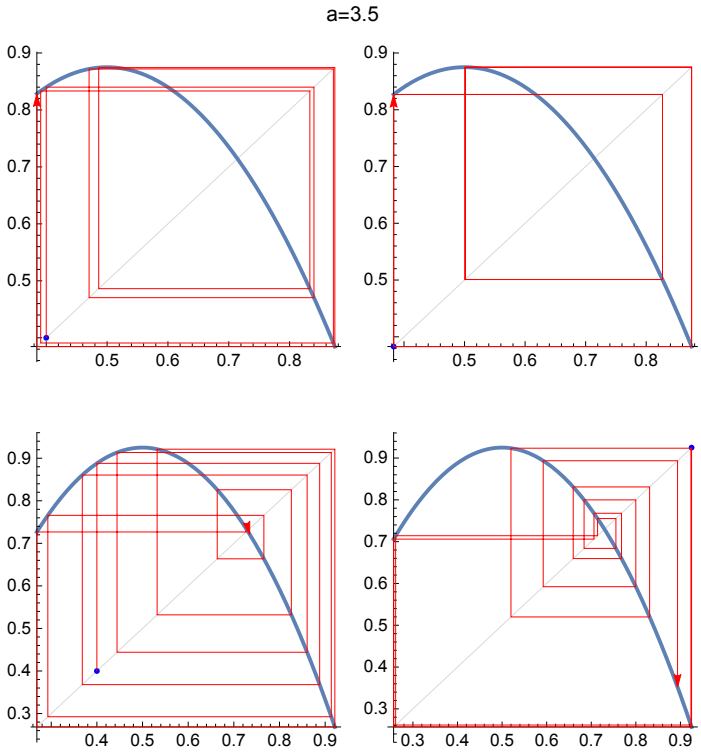
```
Clear[pts, min, max, f, x, x0]
cobwebPoints[f_, x0_, niter_] := Module[{x = x0},
  Table[{{x, x}, {x, x = f[x]}}, {niter}] ~Flatten~ 1
 ]
cobwebPlot[f_, x0_, niter_, burn_: 0] := Module[{pts, min, max},
  pts = cobwebPoints[f, x0, niter + burn];
  min = Min@Flatten@pts;
  max = Max@Flatten@pts;
  Plot[f[x], {x, min, max},
   PlotStyle → {Thick},
   PlotRange → All,
   AxesOrigin → {min, min},
   AspectRatio → 1,
   Prolog → {Thin, LightGray, Line[{{min, min}, {max, max}}]},
   Epilog →
    {{Blue, Point[pts[[2 * burn + 1]]]}, Thin, Red, Arrow[pts[[2 * burn + 1 ;;]]]}
  ]]
```

```
GraphicsRow[
 {cobwebPlot[3.2 # (1 - #) &, 0.4, 9], cobwebPlot[3.2 # (1 - #) &, 0.4, 9, 20]},
 PlotLabel → "a=3.2"]
GraphicsRow[
 {cobwebPlot[3.0 # (1 - #) &, 0.4, 9], cobwebPlot[3.0 # (1 - #) &, 0.4, 9, 200]},
 PlotLabel → "a=3.0"]

GraphicsRow[
 {cobwebPlot[3.5 # (1 - #) &, 0.4, 9], cobwebPlot[3.5 # (1 - #) &, 0.4, 9, 100]},
 PlotLabel → "a=3.5"]
GraphicsRow[
 {cobwebPlot[3.7 # (1 - #) &, 0.4, 15], cobwebPlot[3.7 # (1 - #) &, 0.4, 15, 100]}]
```







Difference Equations

Function Iteration

Nest and NestList

We can produce the values of a recurrence relation through function iteration. For a function of one argument, *Mathematica* makes it easy with its Nest command.

For example, if you would like to repeatedly apply a function *f* starting from an inital value x0,

```
Clear[f];
Nest[f, x0, 3]
```

```
f[f[f[x0]]]
```

If you need the sequence of values from the function iteration, use NestList.

```
NestList[f, x0, 3]
```

```
{x0, f[x0], f[f[x0]], f[f[f[x0]]]}
```

For example, here is a simple recurrence relation computation:

```
Nest[# / 2 &, 128, 3]
```

```
NestList[# / 2 &, 128, 3]
```

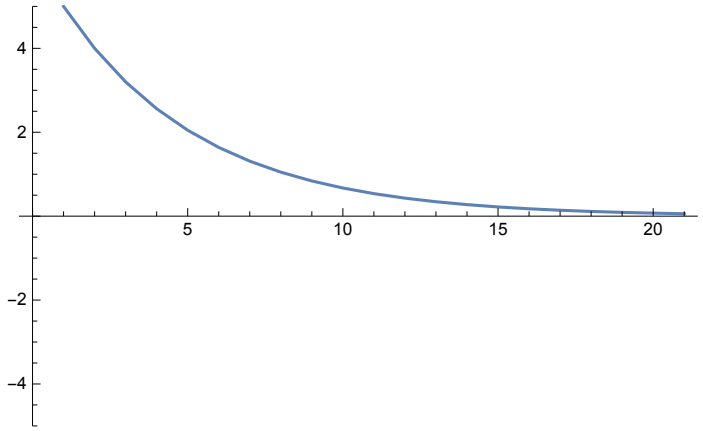
```
16
```

```
{128, 64, 32, 16}
```

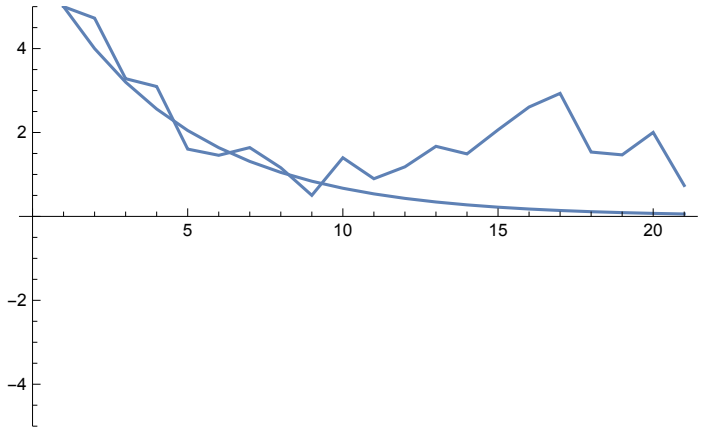
Simulating an AR(1) Process

```
Clear[f, x]
f[x_] := 0.8 * x
niter = 20
series = NestList[f, 5, niter];
plot1 = ListLinePlot[series, PlotRange → {-5, 5}]
```

```
20
```




```
Clear[f, x]
f[x_] := 0.8 * x + RandomReal[{-1, 1}]
series = NestList[f, 5, niter];
plot2 = ListLinePlot[series, PlotRange → {-5, 5}];
Show[{plot1, plot2}]
```



NestWhile and NestWhileList

Often we do not know ahead of time how many iterations we need to do. In this case we can use NestWhile (or NestWhileList). Note that we produce a new value after each time the test passes.

```
NestWhileList[#/2 &, 128, # > 1 &]
```

```
{128, 64, 32, 16, 8, 4, 2, 1}
```

Recursive Definition

We can recursively define the values produced by a difference equation, as long as we provide a base case.

```
Clear[s, t]
```

$$s[t\_Integer] := \begin{cases} 128 & t == 0 \\ \frac{s[t-1]}{2} & t > 0 \end{cases}$$

```
s[3]
```

```
? s
```

```
16
```

Info-dd46dcc0-9bfd-4004-a1f4-7e51dd81645c

```
Global`s
```

Info-dd46dcc0-9bfd-4004-a1f4-7e51dd81645c

$$s[t\_Integer] := \text{Piecewise}\left[\left\{\{128, t == 0\}, \left\{\frac{1}{2} s[t-1], t > 0\right\}\right\}\right]$$

```
Trace[s[3]]
```

$$\left\{s[3], \begin{cases} 128 & 3 == 0 \\ \frac{1}{2} s[3-1] & 3 > 0 \end{cases}, 16\right\}$$

Note the two different uses of the equal sign in this function defintion: delayed assignment (:=), and equality (==). Note that *Mathematica* makes memoization of computed values trivial. If we assign values as we compute them, they become part of the definition of the function s (and correspondingly consume memory). This can substantially increase speed of the some inputs are used many times.

```
Clear[s, t]
```

$$s[t\_Integer] := \begin{cases} s[t] = 128 & t == 0 \\ s[t] = \frac{s[t-1]}{2} & t > 0 \end{cases}$$

Note the three different uses of the equal sign in this function defintion: assignment (=), delayed assignment (:=), and equality (==). Now calling the function s fills a hash table with values, so if we call it again with the same value it is found rather than computed. We can see this by using the Trace function, which will trace the steps in the computation.

```
Trace[s[3]]
```

```
Trace[s[3]]
```

$$\left\{s[3],\ \begin{cases} s[3] = 128 & 3 == 0 \\ s[3] = \frac{1}{2}\, s[3-1] & 3 > 0 \end{cases},\right.$$

$$\left\{\left\{\left\{\{3-1,\ 2\},\ s[2],\ \begin{cases} s[2] = 128 & 2 == 0 \\ s[2] = \frac{1}{2}\, s[2-1] & 2 > 0 \end{cases},\right.\right.\right.$$

$$\left\{\left\{\left\{\{2-1,\ 1\},\ s[1],\ \begin{cases} s[1] = 128 & 1 == 0 \\ s[1] = \frac{1}{2}\, s[1-1] & 1 > 0 \end{cases},\right.\right.\right.$$

$$\left\{\left\{\left\{\{1-1,\ 0\},\ s[0],\ \begin{cases} s[0] = 128 & 0 == 0 \\ s[0] = \frac{1}{2}\, s[0-1] & 0 > 0 \end{cases},\right.\right.\right.$$

$$\left.\left.\left.\{s[0] = 128,\ 128\}\right\},\ 128\right\},\ \left\{\frac{1}{2},\ \frac{1}{2}\right\},\ \frac{128}{2},\ 64\right\},\ s[1] = 64,\ 64\Big\},\ 64\Big\},\ \left\{\frac{1}{2},\ \frac{1}{2}\right\},$$

$$\left.\left.\frac{64}{2},\ 32\right\},\ s[2] = 32,\ 32\right\},\ 32\Big\},\ \left\{\frac{1}{2},\ \frac{1}{2}\right\},\ \frac{32}{2},\ 16\Big\},\ s[3] = 16,\ 16\Big\},\ 16\Big\}$$

```
{s[3], 16}
```

```
Clear[λ]
mA = {{3, 2}, {3, 4}}
mA2 = mA - λ * IdentityMatrix[2]
cpoly = Det[mA2] == 0
cvals = Solve[cpoly, λ]
Solve[mA2.{{x1}, {x2}} == 0]
sys = Eigensystem[mA]
{v1, v2} = sys[[2]]
mA2.v1 /. {λ → sys[[1]][[1]]}
mA2.v2 /. {λ → sys[[1]][[2]]}
P = Transpose@sys[[2]]
Inverse[P].mA.P
```

```
{{3, 2}, {3, 4}}
```

$$\{\{3-\lambda,\ 2\},\ \{3,\ 4-\lambda\}\}$$

$$6 - 7\,\lambda + \lambda^2 == 0$$

$$\{\{\lambda \to 1\},\ \{\lambda \to 6\}\}$$

$$\left\{\{x1 \to 0,\ x2 \to 0\},\ \{x2 \to -x1,\ \lambda \to 1\},\ \left\{x2 \to \frac{3\,x1}{2},\ \lambda \to 6\right\}\right\}$$

```
{{6, 1}, {{2, 3}, {-1, 1}}}
```

```
{{2, 3}, {-1, 1}}
```

```
{0, 0}
```

```
{0, 0}
```

```
{{2, -1}, {3, 1}}
```

```
{{6, 0}, {0, 1}}
```

Recurrence Table

Mathematica provides the RecurrenceTable command to allow easy computation of the values of a difference equation, give initial conditions. Note that the syntax uses the equals symbol, not the assignment symbol.

```
Clear[d]
RecurrenceTable[{d[n + 1] == d[n] / 2, d[0] == 128}, d, {n, 0, 10}]
```

$$\left\{128, 64, 32, 16, 8, 4, 2, 1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}\right\}$$

A Quirk

The RecurrenceTable has a substantial quirk: it pre-evaluates the right hand side functions defining the recurrence realtions.

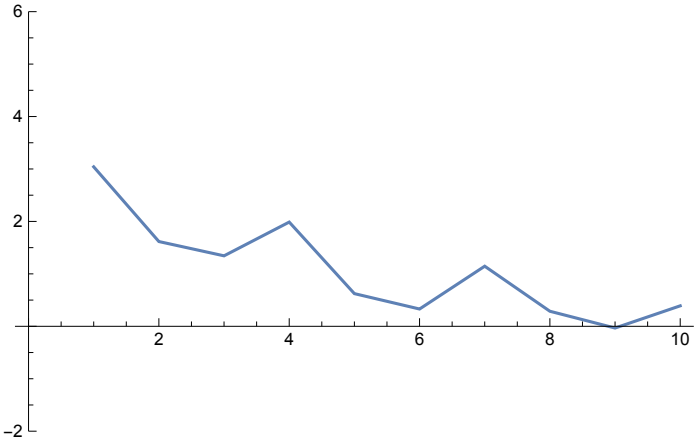
```
Clear[a, n]
(* the following will lead to RandomReal only being called once!! *)
series = RecurrenceTable[
    {a[n + 1] == 0.8 * a[n] + RandomReal[{-1, 1}], a[0] == 5}, a, {n, 1, 10}];
series[[2 ;;]] - 0.8 * series[[ ;; -2]]
```

```
{-0.084371, -0.084371, -0.084371, -0.084371,
 -0.084371, -0.084371, -0.084371, -0.084371, -0.084371}
```

A workaround is to require a numeric input by wrapping the function. (Thanks to Daniel Lichtblau.)

```
Clear[a, n, ct]
shock[n_?IntegerQ] := RandomReal[{-1, 1}]
series = RecurrenceTable[{a[n + 1] == 0.8 * a[n] + shock[n], a[0] == 5}, a, {n, 1, 10}]
ListLinePlot[series, PlotRange → {-2, 6}]
```

```
{3.04071, 1.61645, 1.34378, 1.98742, 0.623257,
 0.329599, 1.14613, 0.28714, -0.0329157, 0.390409}
```



The RecurrenceTable command can handle systems of equations.

```
10
```

```
? RSolve
```

Info-a083cc70-2646-4c7c-9700-315c0574b34a

RSolve[*eqn*, *a*[*n*], *n*] solves a recurrence equation for *a*[*n*].

RSolve[{*eqn*$_1$, *eqn*$_2$, …}, {*a*$_1$[*n*], *a*$_2$[*n*], …}, *n*] solves a system of recurrence equations.

RSolve[*eqn*, *a*[*n*$_1$, *n*$_2$, …], {*n*$_1$, *n*$_2$, …}] solves a partial recurrence equation. ≫

The RSolve command can compute a solution to a difference equation (given initial values):

```
RSolve[{a[n + 1] == 0.8 * a[n], a[0] == 1}, a[n], n]
```

$$\{\{a[n] \to 1. \times 1.25^{-1. n}\}\}$$

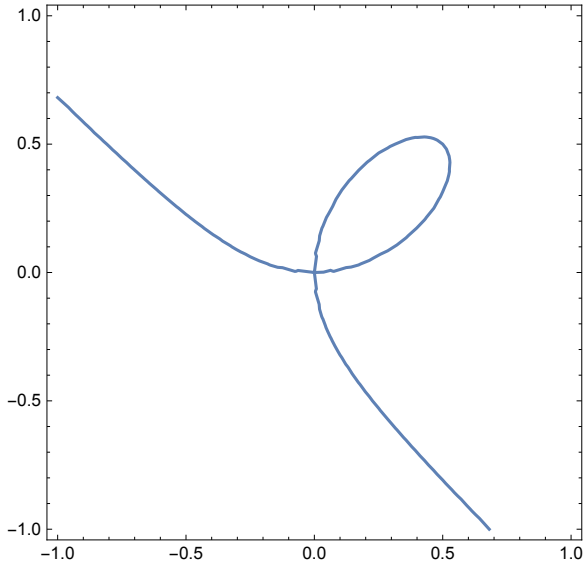
```
RSolve[{a[n + 1] == r * a[n] + 1, a[0] == 1}, a[n], n]
```

$$\left\{\left\{a[n] \to \frac{-1 + r^{1+n}}{-1 + r}\right\}\right\}$$

Implicit Functions

Under suitable conditions, the expression $g(x, y) = 0$ implicitly defines y as a function of x (at least locally). However it may not be possible to determine an explicit function f such that $y = f(x)$. Nevertheless, we can analyze the implicit relationship. For example, we can use ContourPlot (which replaces the old ImplicitPlot) to plot the implicit relationship.

```
ContourPlot[x^3 + y^3 == x y, {x, -1, 1}, {y, -1, 1}, ImageSize → 300]
```



```
eqn = x^3 - 5 x^2 == 2 - 6 x
solns = Solve[eqn, x]
```

$$-5\,x^2 + x^3 == 2 - 6\,x$$

$$\left\{\{x \to 1\},\ \left\{x \to 2 - \sqrt{2}\right\},\ \left\{x \to 2 + \sqrt{2}\right\}\right\}$$

```
eq02 = x^3 + y^3 == x * y && y == x
solns02 = Solve[eq02, {x, y}]
```

$$x^3 + y^3 == x\,y \;\&\&\; y == x$$

$$\left\{\{x \to 0,\ y \to 0\},\ \left\{x \to \frac{1}{2},\ y \to \frac{1}{2}\right\}\right\}$$

```
? Solve
```

Info-55387a0c-7bb1-4c92-87ae-107e083c80ad

Solve[*expr*, *vars*] attempts to solve the system *expr* of equations or inequalities for the variables *vars*.

Solve[*expr*, *vars*, *dom*] solves over the domain *dom*. Common choices of *dom* are Reals, Integers, and Complexes. ≫

```
y = 10 - (x - 5)^2
y' = D[y, x]
FindRoot[y' == 0, {x, 4}]
xmax = x /. %
ymax = y /. %%
```

$$10 - (-5 + x)^2$$

$$-2\,(-5 + x)$$

$$\{x \to 5.\}$$

$$5.$$

$$10.$$

```
? FindRoot
```

Info-2c8ed1fa-2c15-4901-8123-62edcfd33175

FindRoot[f, {x, x_0}] searches for a numerical root of f, starting from the point $x = x_0$.

FindRoot[*lhs* == *rhs*, {x, x_0}] searches for a numerical solution to the equation *lhs* == *rhs*.

FindRoot[{f_1, f_2, …}, {{x, x_0}, {y, y_0}, …}] searches for a simultaneous numerical root of all the f_i.

FindRoot[{eqn_1, eqn_2, …}, {{x, x_0}, {y, y_0}, …}] searches for a numerical solution to the simultaneous equations eqn_i. ≫

Integration

Mathematica can readily find many antiderivatives with the Integrate command. It returns the antiderivative with a zero constant of integration.

```
Integrate[1 / x, x]
```

```
Log[x]
```

It is even possible to include undefined symbols in your integrand:

```
Integrate[x^n, x]
```

$$\frac{x^{1+n}}{1+n}$$

You can give your integrals a more traditional look by using the int and dd keyboard shortcuts.

$$\int x^n \, dx$$

$$\frac{x^{1+n}}{1+n}$$

Mathematica returns this result even though it fails at $n = -1$. This is intentional, but it means you cannot treat the answer mechanically.

If you use this traditional format and your integrand has multiple terms, you will need to put it in parentheses, like this:

$$\int (1 - x) \, dx$$

$$x - \frac{x^2}{2}$$

The integrate command does not isolate its arguments from global assignments. It is good practice to Clear your integration variables before attempting an integration.

```
x = 3; Integrate[x, x]
Clear[x]; Integrate[x, x]
```

```
Integrate::ilim : Invalid integration variable or limit(s) in 3. >>
```

$$\int 3 \, d3$$

$$\frac{x^2}{2}$$

Remember that, by default, variables take on complex values in *Mathematica* so the results sometimes look “needlessly” complex. When appropriate, you may be able to produce simpler results by restricting your variables to the real numbers. Here is an example from Torrence and Torrence (p.224), which reflects *Mathematica*’s choice of principle square root.

```
Integrate[Sqrt[(1 + x^2)^2], x]
Integrate[Sqrt[(1 + x^2)^2], x, Assumptions → x ∈ Reals]
```

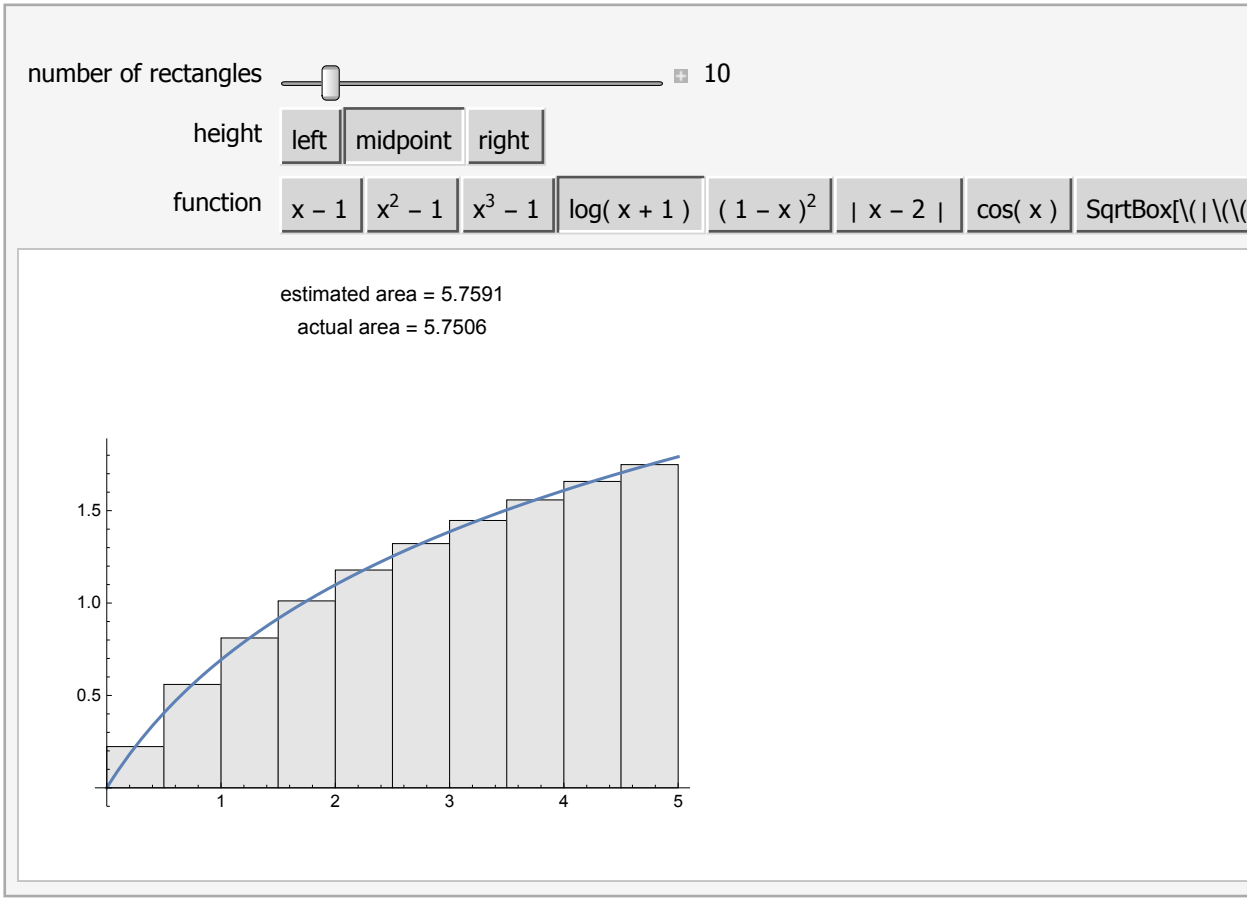
$$\frac{x\sqrt{\left(1+x^2\right)^2}\ \left(3+x^2\right)}{3\left(1+x^2\right)}$$

$$x+\frac{x^3}{3}$$

Riemann Sums

```
fleft[f_, {x_, a_, b_}] := N[f /. x → a];
fmid[f_, {x_, a_, b_}] := N[f /. x → (a + b) / 2];
fright[f_, {x_, a_, b_}] := N[f /. x → b];
Sample[f_, {x_, xl_, xr_}, type_] :=
  {fleft[f, {x, xl, xr}], fmid[f, {x, xl, xr}], fright[f, {x, xl, xr}]}[[type]];
rectCoords[a_, b_, h_] := {{a, 0}, {b, h}};
IsReal[x_] := Module[{}, If[! NumericQ[x] || Im[x] ≠ 0,
    Throw["One or more samples are outside the domain."]];
   x];
FunctionF[x_] := {x - 1, x^2 - 1, x^3 - 1, Log[x + 1],
   (1 - x)^2, Abs[x - 2], Cos[x], Sqrt[Abs[x - 1]]};
ClearAll[FunctionF]
FunctionF = Function[x,
   {x - 1, x^2 - 1, x^3 - 1, Log[x + 1], (1 - x)^2, Abs[x - 2], Cos[x], Sqrt[Abs[x - 1]]}];
FunctionText = {"x - 1", "x^2 - 1", "x^3 - 1", "log( x + 1 )", "( 1 - x )^2",
   "| x - 2 |", "cos( x )", "SqrtBox[\(|\(\(\\)\(x\)\)-1|\)]"};
FunctionButtons = Map[#[[1]] → #[[2]] &,
   Transpose[{Range[Length[FunctionText]], FunctionText}]];
```

```
(* based on code by Ed Pegg http://demonstrations.wolfram.com/RiemannSums/ *)
riemannBlocks[f_, {x_, a_, b_, n_}, height_] := Plot[f, {x, a, b}, Prolog → Table[
    (({{GrayLevel[0.9], EdgeForm[Thin], Rectangle @@ #1}} &)[xl = a + i * ((b - a) / n);
     xr = xl + (b - a) / n;
     (rectCoords[xl, xr, Sample[f, {x, xl, xr}, height]])],
    {i, 0, n - 1}],
  ImagePadding → {{25, 25}, {25, 50}},
  PlotLabel →
   "estimated area = " <> ToString[NumberForm[(b - a) * (Sum[IsReal[Sample[f,
            {x, a + i * ((b - a) / n), a + (i + 1) * ((b - a) / n)}, height]], {i, 0, n - 1}] /
        n), {7, 4}, NumberPadding → {"", "0"}]]
    <> "\n" <> "actual area = " <> ToString[NumberForm[
      Check[Chop[NIntegrate[f, {x, a, b}, AccuracyGoal → 12]], I],
      {7, 4}, NumberPadding → {"", "0"}]]]
Manipulate[riemannBlocks[FunctionF[x][[$idf]], {x, 0, 5, blocks}, type],
 {{blocks, 10, "number of rectangles"}, 4, 70, 1, Appearance → "Labeled"},
 {{type, 2, "height"}, {1 → "left", 2 → "midpoint", 3 → "right"}},
 {{$idf, 4, "function"}, FunctionButtons, ControlType → Setter},
 SaveDefinitions → True]
```



Differential Equations

Univariate Dyanamics

Analytical Solution

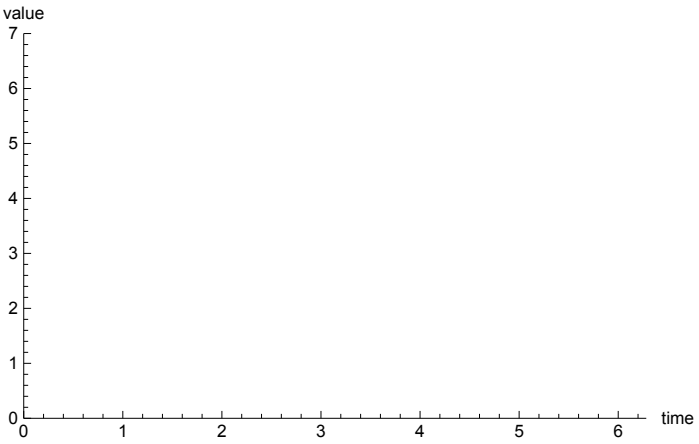
```
DSolve[{y'[x] == α y[x], y[0] == 1}, y[x], x]
```

DSolve::ivar : $10-(-5+x)^2$ is not a valid variable. >>

```
DSolve[{(-2 (-5 + x)) [x] == α (10 - (-5 + x)^2) [x], (10 - (-5 + x)^2) [0] == 1},
 (10 - (-5 + x)^2) [x], x]
```

Graphical Illustration

```
(* provide an empty list to hold points *)
pts = {};
(* function for updating pts *)
ptupdate[t_, y_] := (AppendTo[pts, {t, y}]; Pause[0.05])
(* use Dynamic so pt updating causes ListPlot to revaluate *)
Dynamic[
 ListPlot[pts, PlotRange → {{0, 2 * Pi}, {0, 7}}, AxesLabel → {"time", "value"}]]
(* our ODE *)
eq = y''[t] == Cos[t];
(* solve our ode,use EvaluationMonitor to catch points *)
sol = NDSolve[{eq, y[0] == 1, y'[0] == 1},
   y[t], {t, 0, 2 * Pi}, EvaluationMonitor :↦ ptupdate[t, y[t]]];
```



NDSolve::dsfun : $(10-(-5+x)^2)$[t] cannot be used as a function. >>

Coupled System

Consider the coupled system of differential equations

$\dot{x}_1 = a_{11}\, x_1 + a_{12}\, x_2$

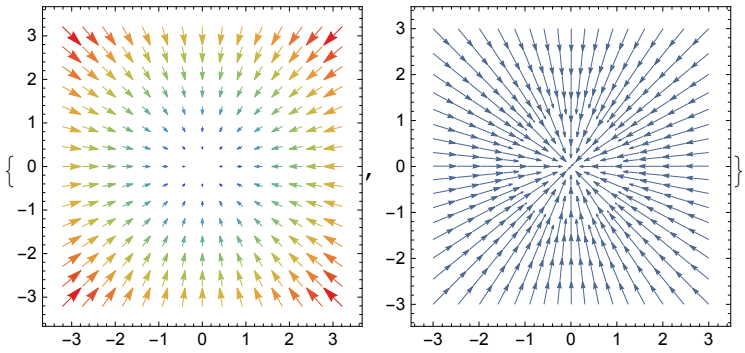
$\dot{x}_2 = a_{21}\, x_1 + a_{22}\, x_2$

We can write this as

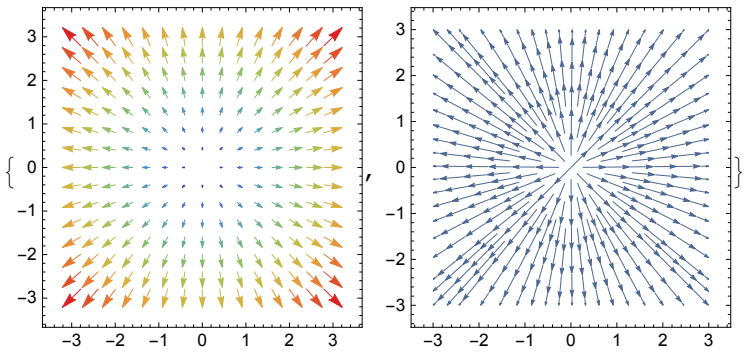
$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

or $\dot{x} = A\, x$ where a_{ij} are the constant coefficients of a matrix A.
Naturally, the behavior of this system depends on the coefficients. Here are some illustrations.

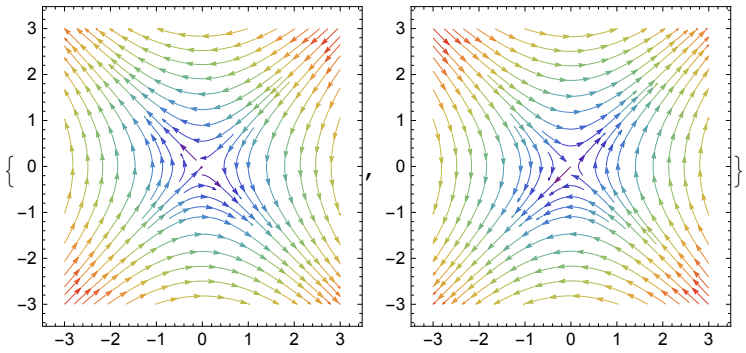
```
Clear[x1, x2]
(* a stable node (vector and stream plot) *)
{VectorPlot[{{-1, 0}, {0, -1}}.{x1, x2}, {x1, -3, 3}, {x2, -3, 3},
  VectorColorFunction → "Rainbow", ImageSize → Small],
 StreamPlot[{{-1, 0}, {0, -1}}.{x1, x2},
  {x1, -3, 3}, {x2, -3, 3}, ImageSize → Small]}
```



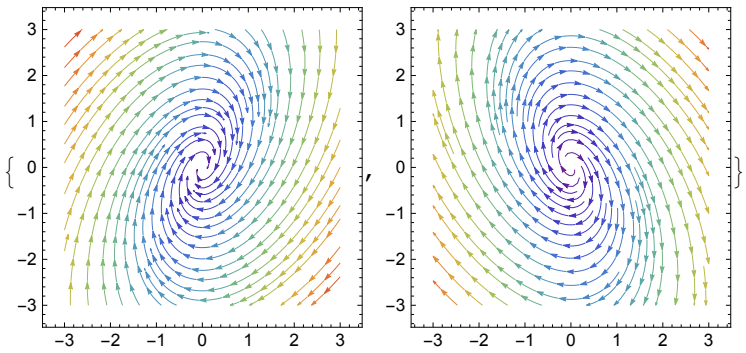
```
(* an unstable node (vector and stream plot) *)
{VectorPlot[{{1, 0}, {0, 1}}.{x1, x2}, {x1, -3, 3}, {x2, -3, 3},
  VectorColorFunction → "Rainbow", ImageSize → Small],
 StreamPlot[{{1, 0}, {0, 1}}.{x1, x2}, {x1, -3, 3}, {x2, -3, 3}, ImageSize → Small]}
```



```
(* saddle points *)
{
 StreamPlot[{{0, -1}, {-1, 0}}.{x1, x2}, {x1, -3, 3}, {x2, -3, 3},
  StreamColorFunction → "Rainbow", StreamStyle → {Thin}, ImageSize → Small],
 StreamPlot[{{0, 1}, {1, 0}}.{x1, x2}, {x1, -3, 3}, {x2, -3, 3},
  StreamColorFunction → "Rainbow", StreamStyle → {Thin}, ImageSize → Small]
}
```



```
(* stable and unstable focus points ("spiral points") *)
{
 StreamPlot[{{-0.5, 0.5}, {-1, 0}}.{x1, x2}, {x1, -3, 3}, {x2, -3, 3},
  StreamColorFunction → "Rainbow", StreamStyle → {Thin}, ImageSize → Small],
 StreamPlot[{{0.5, 0.5}, {-1, 0}}.{x1, x2}, {x1, -3, 3}, {x2, -3, 3},
  StreamColorFunction → "Rainbow", StreamStyle → {Thin}, ImageSize → Small]
}
```



Here is an equivalent representation of the system, using differential operator notation.

$$\begin{pmatrix} a_{11} - D & a_{12} \\ a_{21} & a_{22} - D \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 0$$

Recall that the eigenvalues λ_1 and λ_2 of A are the roots of the quadratic equation $\det(A - \lambda I) = 0$ and the corresponding eigenvectors v solve the equation $(A - \lambda I)\, v = 0$.

$$\begin{pmatrix} a_{11} - \lambda & a_{12} \\ a_{21} & a_{22} - \lambda \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = 0$$

Note the parallels in the previous two equations!

```
charpoly = Det[{{a11 - λ, a12}, {a21, a22 - λ}}];
Collect[charpoly, λ]
Solve[charpoly == 0, λ]
```

$$-\text{a12 a21} + \text{a11 a22} + (-\text{a11} - \text{a22})\ \lambda + \lambda^2$$

$$\left\{\left\{\lambda \to \frac{1}{2}\left(\text{a11} + \text{a22} - \sqrt{\left(\text{a11}^2 + 4\ \text{a12 a21} - 2\ \text{a11 a22} + \text{a22}^2\right)}\right)\right\},\right.$$
$$\left.\left\{\lambda \to \frac{1}{2}\left(\text{a11} + \text{a22} + \sqrt{\left(\text{a11}^2 + 4\ \text{a12 a21} - 2\ \text{a11 a22} + \text{a22}^2\right)}\right)\right\}\right\}$$

Depending on the discriminant, the λ can be real or complex. The resulting solution will have the form $x = e^{\lambda_1 t} v^1 + e^{\lambda_2 t} v^2$ where λ_i are the eigenvalues of the systems and v^i are the corresponding eigenvectors.

```
Eigenvectors[{{a11, a12}, {a21, a22}}]
```

$$\left\{\left\{-\frac{1}{2\ \text{a21}}\left(-\text{a11} + \text{a22} + \sqrt{\left(\text{a11}^2 + 4\ \text{a12 a21} - 2\ \text{a11 a22} + \text{a22}^2\right)}\right), 1\right\},$$
$$\left.\left\{-\frac{1}{2\ \text{a21}}\left(-\text{a11} + \text{a22} - \sqrt{\left(\text{a11}^2 + 4\ \text{a12 a21} - 2\ \text{a11 a22} + \text{a22}^2\right)}\right), 1\right\}\right\}$$

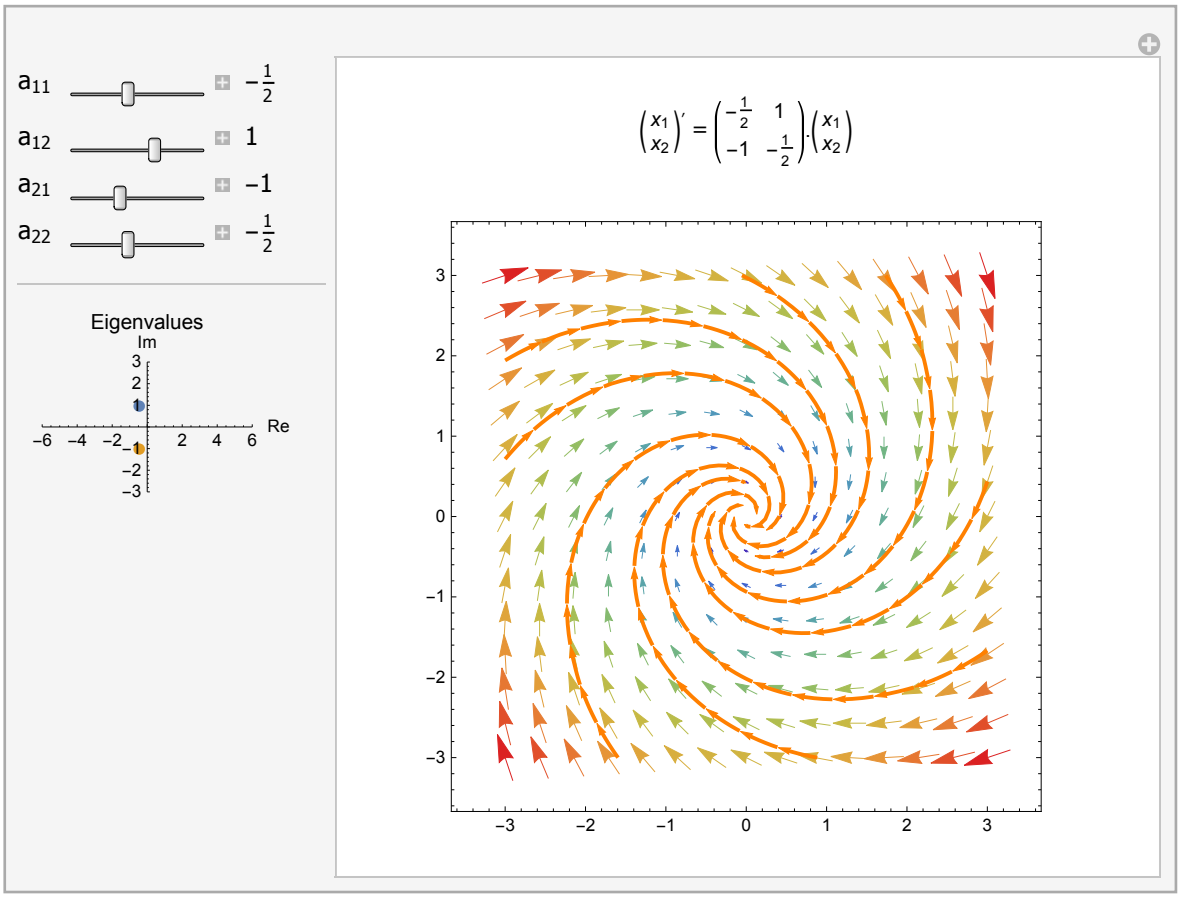
We can get the eigensystem of eigenvalues and eigenvectors all at one go.

```
Eigensystem[{{a11, a12}, {a21, a22}}]
```

$$\left\{\left\{\frac{1}{2}\left(\text{a11} + \text{a22} - \sqrt{\left(\text{a11}^2 + 4\ \text{a12 a21} - 2\ \text{a11 a22} + \text{a22}^2\right)}\right),\right.$$
$$\left.\frac{1}{2}\left(\text{a11} + \text{a22} + \sqrt{\left(\text{a11}^2 + 4\ \text{a12 a21} - 2\ \text{a11 a22} + \text{a22}^2\right)}\right)\right\},$$
$$\left\{\left\{-\frac{1}{2\ \text{a21}}\left(-\text{a11} + \text{a22} + \sqrt{\left(\text{a11}^2 + 4\ \text{a12 a21} - 2\ \text{a11 a22} + \text{a22}^2\right)}\right), 1\right\},\right.$$
$$\left.\left.\left\{-\frac{1}{2\ \text{a21}}\left(-\text{a11} + \text{a22} - \sqrt{\left(\text{a11}^2 + 4\ \text{a12 a21} - 2\ \text{a11 a22} + \text{a22}^2\right)}\right), 1\right\}\right\}\right\}$$

Let us plot the system's direction field and phase portrait. In this example, you can adjust the constants in the equations to discover both real and complex solutions. Using Euler's formula $e^{i t} = \cos(t) + i \sin(t)$, the solutions take the form $\vec{x} = c_1\, u(t) + c_2\, w(t)$. Since the Wronskian is never zero, it follows that $u(t)$ and $w(t)$ constitute a fundamental set of (real-valued) solutions to the system of equations.

```
Manipulate[(* Based on a Mathematica demonstration by Steven Wilkerson. *)
 Module[{ae, vals, vecs, vecplot, streamplot},
  ae = {{a11n, a12n}, {a21n, a22n}};
  {vals, vecs} = Eigensystem[ae];
  lplot = ListLinePlot[
    {{{Re[vals[[1]]], Im[vals[[1]]]}}, {{Re[vals[[2]]], Im[vals[[2]]]}}},
    PlotMarkers → {"●", "●"}, PlotLabel → "Eigenvalues", AxesLabel → {"Re", "Im"},
    AxesOrigin → Automatic, PlotRange → {{-6, 6}, {-3, 3}}, ImageSize → {150, 100}];
  vecplot = VectorPlot[ae.{x1, x2}, {x1, -3., 3.}, {x2, -3., 3.},
    VectorPoints → ControlActive[4, 15], VectorColorFunction → "Rainbow"];
  streamplot = StreamPlot[ae.{x1, x2}, {x1, -3, 3}, {x2, -3, 3}, Axes → True,
    StreamStyle → {Thick, Orange}, StreamPoints → ControlActive[2, 8]];
  Show[{vecplot, streamplot}, ImageSize → {400, 400}, ImagePadding → 20,
   PlotLabel → Pane[Style[HoldForm[{{x1}, {x2}}' == {{a11, a12}, {a21, a22}}.{{x1}, {x2}}] /.
       {a11 → a11n, a12 → a12n, a21 → a21n, a22 → a22n}],
     ImageSize → {400, 50}, Alignment → Center]]],
 {{a11n, -1/2, "a11"}, -3, 3, 1/4, Appearance → "Labeled", ImageSize → Tiny},
 {{a12n, 1, "a12"}, -3, 3, 1/4, Appearance → "Labeled", ImageSize → Tiny},
 {{a21n, -1, "a21"}, -3, 3, 1/4, Appearance → "Labeled", ImageSize → Tiny},
 {{a22n, -1/2, "a22"}, -3, 3, 1/4, Appearance → "Labeled", ImageSize → Tiny},
 Delimiter,
 Dynamic[Show[lplot]],
 ControlPlacement → Left, Alignment → Center,
 TrackedSymbols :→ {a11n, a12n, a21n, a22n},
 SynchronousUpdating → False, ContinuousAction → False]
```



Solving Equations

The output from an equation solution may initially look needlessly complicated.

```
soln01 = Solve[x + 2 == 4]
soln02 = Solve[x^2 - x == 2]
```

```
{{x → 2}}
```

```
{{x → -1}, {x → 2}}
```

```
x /. soln02
```

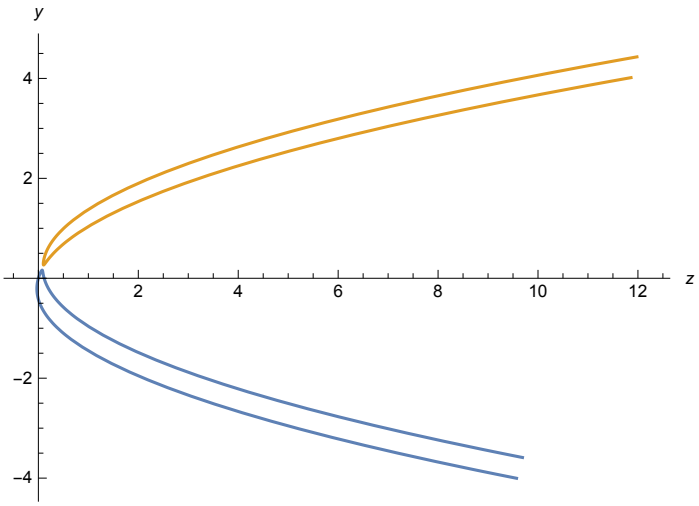
```
{-1, 2}
```

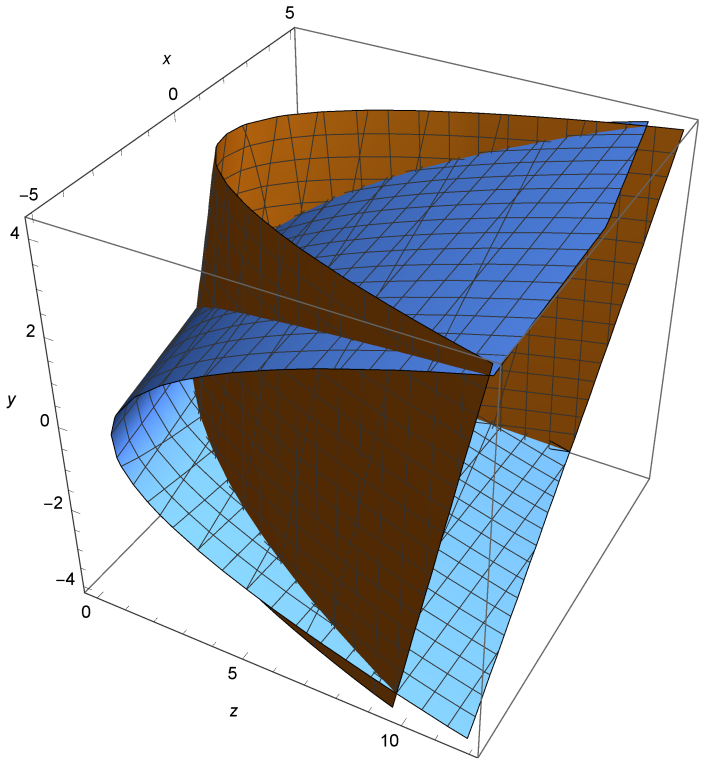
```
Clear[x, y, z] (* example by Bob Hanlon *)
f[x_, y_, z_] := 3 x^2 + 2 y - 7 z;
g[x_, y_, z_] := 2 x - 6 y^2 + 9 z;
eqns = {f[x, y, z] == 0, g[x, y, z] == 0};
soln = Solve[eqns, {y, z}]
```

```
ParametricPlot[Evaluate[{z, y} /. soln], {x, -5, 5}, AxesLabel → {z, y}]
```

```
ContourPlot3D[Evaluate[eqns], {z, -.2, 12},
 {x, -5, 5}, {y, -4, 4.25}, AxesLabel → {z, x, y}]
```

$$\left\{\left\{y \to \frac{1}{42}\left(9 - \sqrt{3}\ \sqrt{27 + 196\,x + 378\,x^2}\right),\ z \to \frac{1}{147}\left(9 + 63\,x^2 - \sqrt{3}\ \sqrt{27 + 196\,x + 378\,x^2}\right)\right\},\right.$$
$$\left.\left\{y \to \frac{1}{42}\left(9 + \sqrt{3}\ \sqrt{27 + 196\,x + 378\,x^2}\right),\ z \to \frac{1}{147}\left(9 + 63\,x^2 + \sqrt{3}\ \sqrt{27 + 196\,x + 378\,x^2}\right)\right\}\right\}$$





Linear Algebra

Gaussian Elimination

We begin with a system Ax = *b*, where we wish to solve for *x*.

```
mA = {{1, 2, 3}, {2, 2, 3}, {3, 2, 1}};
mb = {{14}, {15}, {10}};
(mA // MatrixForm).x == (mb // MatrixForm)
mx = {{1}, {2}, {3}}
mA.mx
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}.x == \begin{pmatrix} 14 \\ 15 \\ 10 \end{pmatrix}$$

```
{{1}, {2}, {3}}
```

```
{{14}, {15}, {10}}
```

Create the “augmented matrix” [*A* | *b*].

```
mAx = ArrayFlatten[{{mA, mb}}];
mAx // MatrixForm
```

$$\begin{pmatrix} 1 & 2 & 3 & 14 \\ 2 & 2 & 3 & 15 \\ 3 & 2 & 1 & 10 \end{pmatrix}$$

We will consider a manual row reduction.

Eigenvalues and Eigenvectors

Given a matrix A, and eigenvalue is a scalar λ such that the matrix $(A - \lambda I)$ is singular. We can find such a scalar by solving the characteristic equation $|A - \lambda I| = 0$. For example, consider the matrix

$$A = \begin{pmatrix} 4 & -1 \\ -3 & 2 \end{pmatrix} \Rightarrow A - \lambda I = \begin{pmatrix} 4-\lambda & -1 \\ -3 & 2-\lambda \end{pmatrix}$$

The characteristic polynomial is $|A - \lambda I|$, which is $\lambda^2 - 6\lambda + 5$. This is just an ordinary polynomial in λ. the characteristic equation $\lambda^2 - 6\lambda + 5 = 0$. Applying the quadratic equation, we get solutions $\lambda_1, \lambda_2 = 1, 5$. We can use *Mathematica* to implement exactly these steps.

```
mA = {{4, -1}, {-3, 2}}
mL = mA - λ * IdentityMatrix[2]
charpoly = Det[mL]
soln = Solve[charpoly == 0, λ]
```

```
{{4, -1}, {-3, 2}}
```

```
{{4 - λ, -1}, {-3, 2 - λ}}
```

$$5 - 6\lambda + \lambda^2$$

```
{{λ → 1}, {λ → 5}}
```

Mathematica also offers some special commands that allow us to proceed more concisely.

```
CharacteristicPolynomial[mA, λ]
Eigenvalues[mA]
```

$$5 - 6\lambda + \lambda^2$$

```
{5, 1}
```

With each eigenvalue λ of a matrix A, we can associate an eigenvector v, such that

$$Av = \lambda v$$

In other words, premultiplying A times one of its eigenvectors produces the same outcome as scaling that vector by the associated eigenvalue. Clearly if v is an eigenvector, so is any nonzero scalar multiple of v. Eigenvectors are not unique.

```
mL1 = mL /. soln[[1]]; mL2 = mL /. soln[[2]];
Solve[mL1.{{x1}, {x2}} == 0, {x1, x2}]
```

```
Solve::svars : Equations may not give solutions for all "solve" variables. >>
```

```
{{x2 → 3 x1}}
```

```
Solve[mL1.{{x1}, {x2}} == 0, {x1, x2}] /. {x1 → 1}
Solve[mL2.{{x1}, {x2}} == 0, {x1, x2}] /. {x1 → 1}
```

Solve::svars : Equations may not give solutions for all "solve" variables. ≫

```
{{x2 → 3}}
```

Solve::svars : Equations may not give solutions for all "solve" variables. ≫

```
{{x2 → -1}}
```

Once again, *Mathematica* offers some specialized commands for exploring eigensystems. The Eigenvectors command produces eigenvectors (naturally enough), while the Eigensystem command returns both the eigenvalues and associated eigenvectors.

```
Clear[λ]
mB = 2 * {{1, 0, 2}, {0, 5, 0}, {3, 0, 2}};
mB // MatrixForm
cpB = CharacteristicPolynomial[mB, λ]
m1 = mB + 2 * IdentityMatrix[3]
m1 // MatrixForm
Solve[m1.{x1, x2, x3} == 0, {x1, x2, x3}]
Factor[cpB]
Eigensystem[mB]
evec = {{1}, {0}, {-1}}
mB.evec == -1 * evec
```

$$\begin{pmatrix} 2 & 0 & 4 \\ 0 & 10 & 0 \\ 6 & 0 & 4 \end{pmatrix}$$

$$-160 - 44\,\lambda + 16\,\lambda^2 - \lambda^3$$

```
{{4, 0, 4}, {0, 12, 0}, {6, 0, 6}}
```

$$\begin{pmatrix} 4 & 0 & 4 \\ 0 & 12 & 0 \\ 6 & 0 & 6 \end{pmatrix}$$

Solve::svars : Equations may not give solutions for all "solve" variables. ≫

```
{{x2 → 0, x3 → -x1}}
```

$$-(-10+\lambda)\,(-8+\lambda)\,(2+\lambda)$$

```
{{10, 8, -2}, {{0, 1, 0}, {2, 0, 3}, {-1, 0, 1}}}
{{1}, {0}, {-1}}
False
Eigenvectors[mA]
Eigensystem[mA]
{{-1, 1}, {1, 3}}
{{5, 1}, {{-1, 1}, {1, 3}}}
```

```
mP = {{1, 1}, {3, -1}}
Inverse[mP].mA.mP
```

```
{{1, 1}, {3, -1}}
```

```
{{1, 0}, {0, 5}}
```

External Data

Writing CSV Files

We can write a matrix of data to a CSV file with the Export command.

```
mA = RandomInteger[10, {5, 2}];
Export["c:/temp/temp.csv", mA, "CSV", "TableHeadings" → {"col1", "col2"}]
```

```
c:/temp/temp.csv
```

Reading CSV Files

Similarly, data can be read from a CSV file with Import. We can use the “HeaderLines” option to strip initial lines.

```
Import["c:/temp/temp.csv"] // Grid
Import["c:/temp/temp.csv", "HeaderLines" → 1]
```

```
col1  col2
 5     1
 5     2
 3     0
 7     5
 1     7
```

```
{{5, 1}, {5, 2}, {3, 0}, {7, 5}, {1, 7}}
```

We will usually use Import to read external data, but when we have a single series (or complicated formats) we may want to use ReadList..

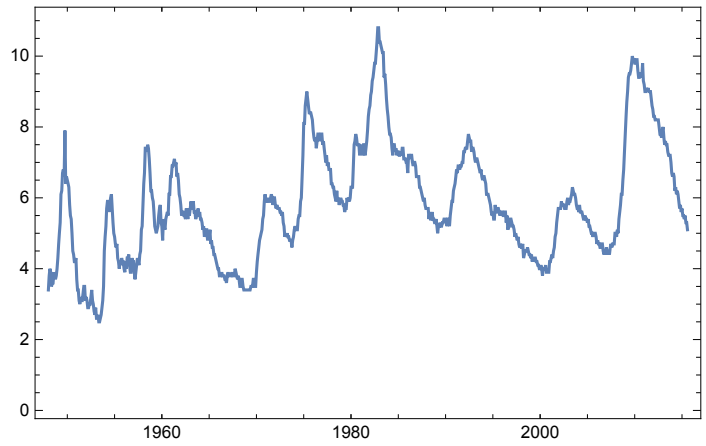
```
ReadList["c:/temp/temp.csv", "Record"]
```

```
{col1,col2, 5,1, 5,2, 3,0, 7,5, 1,7}
```

Data from the Web

```
unrate = Import[
   "http://research.stlouisfed.org/fred2/series/UNRATE/downloaddata/UNRATE.csv"];
Grid[unrate[[1 ;; 10]]]
DateListPlot[Rest[unrate]]
```

DATE	VALUE
1948-01-01	3.4
1948-02-01	3.8
1948-03-01	4.
1948-04-01	3.9
1948-05-01	3.5
1948-06-01	3.6
1948-07-01	3.6
1948-08-01	3.9
1948-09-01	3.8



Programming in Mathematica

Pattern Matching

Mathematica makes heavy use of patterns and pattern matching. Here we provide a basic introduction.

Useful resources:
http://reference.wolfram.com/mathematica/guide/Patterns.html

Basic Patterns

A blank is a pattern that can match any *Mathematica* expression. It can be written as `Blank[]` or as an underscore. `Blank` accepts one optional argument, which we may use to restrict the match by specifying the type of head the expression must have. Use `MatchQ` to see if an expression is matched by a pattern.

```
_ // FullForm
MatchQ[2.5, _]
_Integer // FullForm
MatchQ[2.5, _Integer]
```

```
Blank[]
```

```
True
```

```
Blank[Integer]
```

```
False
```

We can also test for satisfaction of a condition, such as positivity, with `PatternTest`, which from a pattern `p` and a condition `c` produces a pattern that matches and expression when `p` matches and `c` also evaluates the expression to True. (A shorthand is to follow a pattern with a question mark and the test.)

```
_?Positive // FullForm
MatchQ[2.5, _?Positive]
_Integer?Positive // FullForm
MatchQ[2.5, _Integer?Positive]
MatchQ[3, PatternTest[_Integer, x ↦ 0 < x < 5]]
```

```
PatternTest[Blank[], Positive]
```

```
True
```

```
PatternTest[Blank[Integer], Positive]
```

```
False
```

```
True
```

Basic Pattern Matching

The simplest pattern object is `Blank[]`, which matches any expression. *Mathematica* allows a single underscore as a synonym for `Blank[]`, and this convenient shorthand is almost always prefered. We can use `MatchQ` to query whether an expression is matched by a pattern object.

```
MatchQ[what + ever, _]  (* matches, because Blank[] matches any expression *)
```

```
True
```

We can provide a head (e.g., object type) as a single argument to `Blank`; for example, `Blank[List]` or `Blank[Symbol]`.

```
MatchQ[what, _Symbol] (* matches *)
MatchQ[what + ever, _Symbol] (* does NOT match; wrong head *)
MatchQ[what + ever, _Plus]  (* matches; correct head *)
```

```
True

False

True
```

There are many uses of patterns. One use is to select items from a list. The `Cases` command takes a list and a pattern and returns the sublist of items matching the pattern.

```
Clear[a, b]
Cases[{a, b, 1.1, 2.2, 4, 5, Pi}, _Integer]
```

```
{4, 5}
```

We can supplement our pattern matching requirements with a test, using `PatternTest`, for which the question mark is available as an infix shorthand.

```
lt20 = PatternTest[_Integer, x ↦ x < 20]
MatchQ[10, lt20]
MatchQ[30, lt20]
```

```
_Integer?(Function[x, x < 20])

True

False
```

Naturally we can use the `Cases` command with `PatternTest` as well. (In this case, it would be more natural to use `Condition`; see the next section.)

```
lst = Range[30];
Cases[lst, _Integer?(x ↦ x < 20)]
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19}
```

Suppose we want all the elements that do not match a pattern. For that, we can create our pattern with `Except`. For example, to get the sublist of non-prime numbers in a list of integers, we could use `Cases` as follows.

```
Cases[lst, Except[_Integer?(x ↦ x < 20)]]
```

```
{20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30}
```

Naming Pattern Objects

To bind a name to a pattern, use the `Pattern` command, or a colon as an infix shorthand for it. If the pattern is a `Blank[]` even the colon can be omitted. In fact, because the colon operator has very low operator precedence, it can be a good idea to omit it when possible. For example, since `+` has priority

over `:`, the next two patterns are not the same. (We can of course use parentheses.)

```
Pattern[x, _] + Pattern[y, _] // FullForm
x : _ + y : _ // FullForm
(x : _) + (y : _) // FullForm
```

```
Plus[Pattern[x, Blank[]], Pattern[y, Blank[]]]

Optional[Pattern[x, Plus[y, Blank[]]], Blank[]]

Plus[Pattern[x, Blank[]], Pattern[y, Blank[]]]
```

Using `Pattern`, we can name patterns for subsequent reference in an expression. There is also a shortcut notation: just separate the name from the pattern with a colon. In the simplest cases, we are even allowed to omit the colon.

```
Pattern[x, _] // FullForm
x : _ // FullForm
x_ // FullForm
```

```
Pattern[x, Blank[]]

Pattern[x, Blank[]]

Pattern[x, Blank[]]
```

However, in more complex cases, we need to be more explicit.

```
x_Integer?Positive // FullForm (* uh oh ... *)
x : _Integer?Positive // FullForm (* fixed! *)
x : (_Integer?Positive) // FullForm (* fixed! *)
```

```
PatternTest[Pattern[x, Blank[Integer]], Positive]

Pattern[x, PatternTest[Blank[Integer], Positive]]

Pattern[x, PatternTest[Blank[Integer], Positive]]
```

By naming our patterns, we are able to reference them subsequently uses. For example, to conditionally apply a pattern use the `Condition` command or its `/;` infix shorthand.

```
x_ /; x > 0 // FullForm
Map[MatchQ[#, %] &, {-1, 0, 1}]
```

```
Condition[Pattern[x, Blank[]], Greater[x, 0]]

{False, False, True}
```

```
Cases[RandomInteger[{0, 100}, 20], x_ /; x < 20]
```

```
{12, 18, 18, 11, 2}
```

We use patterns with delayed evaluation to define functions, possibly with type checking and default values.

```
ClearAll[f, g, h]
f[x : _] := x * x
f[2.5]
g[x : _Integer : 0] := x * x
g[]
g[2.5]
```

```
6.25
```

```
0
```

```
g[2.5]
```

Note the inclusion of a second colon followed by a default value. Even this simple cases has a subtlety: the default is not applied if the argument does not match.

Pattern-based function definitions can also make use of `Condition`:

```
ClearAll[f, g]
f[x_ /; x > 0] := x * x
Map[f, {-1, 0, 1}]
g[x_] := x * x /; x > 0
Map[g, {-1, 0, 1}]
```

```
{f[-1], f[0], 1}
```

```
{g[-1], g[0], 1}
```

Use `HoldPattern` for Sensible Matching

Even two different `Blank[]` patterns can stand for different things, in Mathematica they evaluate as equal.

```
Blank[] == Blank[]
```

```
True
```

Thus for example, Mathematica is perfectly willing to add or multiply them.

```
{_ + _, _ * _}
```

$$\{2 \_, \_^2\}$$

This creates a problem for pattern matching, which can arise if evaluation of a pattern produces an unexpected pattern. For example,

```
Clear[a, b]
MatchQ[a + b, _ + _]
```

```
False
```

If we use `Trace` we discover why the match failed:

```
Trace[MatchQ[a + b, _ + _]]
```

```
{{_ + _, 2 _}, MatchQ[a + b, 2 _], False}
```

As a solution for such problems, Mathematica provides `HoldPattern`. We use `HoldPattern` to keep our pattern in unevaluated form.

```
MatchQ[a + b, HoldPattern[_ + _]]
```

```
True
```

Rules and Replacement

Mathematica allows the use of pattern matching with replacement rules to transform expressions. A rule is created with the `Rule` command or its right-arrow inflix shorthand.

```
rule20 = Rule[x, 20]
rule30 = x → 30
```

```
x → 20
```

```
x → 30
```

We can apply rules using the `Replace`.

```
Clear[x, y, z]
Replace[x, rule20]
Replace[x, rule30]
```

```
20
```

```
30
```

The simple replace command works on whole expressions, not subexpressions. For example, the following does not produce a replacement.

```
Replace[x + y, x → x1 + x2]
```

```
x + y
```

For this reason, we usually use `ReplaceAll` or its `/.` infix shorthand. This is usually more useful: it will attempt to tranform each subpart of an expression.

```
Clear[x, x1, x2, x, y, z]
ReplaceAll[x + y + z^x, x → x1 + x2]
x + y + z^x /. {x → x1 + x2}
```

```
x1 + x2 + y + z^(x1+x2)
```

```
x1 + x2 + y + z^(x1+x2)
```

Replacement rules can be used to evaluate expressions for particular values of the symbols without assigning new values to the global symbols.

```
x + y + z^x /. {x → 1, y → 2, z → 3}
```

```
6
```

If you wish to set a global value for a varaible *x*, you make an assignment, like *x* = 1. Then *Mathematica* will replace your variable *x* with the value 1 everywhere that it occurs. Sometimes it is more useful to replace *x* just in a particular expression. *Mathematica* lets you do this in a number of ways with fine control. Here we illustrate the most common case: the use of ReplaceAll, with the common slash-dot notation.

```
Clear[x]
x /. x → 1 (* replace x with 1 in this expression only *)
x /. {x → 1, x → 2} (* only the first match will be used *)
x /. {{x → 1}, {x → 2}}
(* a list substitution lists → a list of expression values *)
x /. Table[{x → i}, {i, 1, 10}] (* same idea *)
x (* none of this changes the value of x *)
```

```
1
```

```
1
```

```
{1, 2}
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

```
x
```

Rules for rewriting expressions are fundamental to Mathematica. We create a rule with the `Rule` command or its `->` shorthand. We can then use `ReplaceAll` (or its `/.` infix shorthand) to transform an expression using our rule.

```
Clear[a, b]
rule01 = Rule[a, b]
ReplaceAll[a * a, rule01]
a * a /. rule01
```

```
a → b
```

```
b^2
```

```
b^2
```

When we use `Rule` to create a rule, the replacement expression is evaluated before the rule is created.

```
x = 2
rule02 = a → x
x = 3
a * a /. rule02
```

```
2
```

```
a → 2
```

```
3
```

```
4
```

If we do not want the replacement expression to be evaluated until after the substitution has taken place, we instead use `RuleDelayed` (or its `:>` shorthand).

```
x = 2
rule02 = a :> x
x = 3
a * a /. rule02
```

```
2
```

```
a :→ x
```

```
3
```

```
9
```

Control Flow

Traditional Structures

Branching

As its basic branching construct, Mathematica provides the `If` command, which conditionally returns a value. The `If` command takes three arguments: a boolean test, the expressions to evaluate if the test is True, and the expression to evaluate if the test is False. (Important: the last two arguments are only evaluated as needed.)

```
If[True, "it's true", "it's false"]
If[False, "it's true", "it's false"]
```

```
it's true
```

```
it's false
```

If the third argument is omitted but the test is False, then `If` returns `Null`.

```
FullForm[If[False, "it's true"]]
```

```
Null
```

Unusually, Mathematica also allows a fourth argument, which is returned if the test value is not boolean. Note that in contrast to some other languages, `0` and `1` are not treated as boolean.

```
If[1, 2, 3, 4]
```

```
4
```

A nice example from Torrence and Torrence:

```
fmtTable = Table[If[PrimeQ[n], Style[n, {Bold, Blue}], n], {n, 100}]
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,
 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61,
 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81,
 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100}
```

If you need to conditionally choose between multiple expressions (not just two), Mathematica also provides the `Which` and `Switch` commands.

Looping

The basic looping commands are `Do`, `While`, and `For`. Here we compute 5! as an illustration.

The first argument to Do can be a compound expression, and `Do` can use multiple iterators, but here we stick with the simplest use.

```
fac5 = 1; Do[fac5 *= k, {k, 1, 5}]; fac5
```

```
120
```

It is a very nice feature of `Do` that the iterator can be over any list of values.

```
fac5 = 1; Do[fac5 *= k, {k, Range[5]}]; fac5
```

```
120
```

Note that the value of a compound expression, produced by separating expressions with semicolons, is the value of the last expression. Next, we generalize this into a function.

```
Clear[nfac];
nfac[n_] := (facn = 1; Do[facn *= k, {k, 1, n}]; facn)
nfac[5]
```

```
120
```

The While command repeatedly evaluates its second argument as long as its first argument evaluates to True.

```
n = 5; fac5 = 1; While[n > 0, fac5 *= n--]; fac5
```

```
120
```

The `For` command accepts four arguments: an initialization, a test that controls the loop, an increment statement, and the loop body. Note that in contrast with the `for` statement in C or C++, the arguments to `For` are separated by commas (not semicolons).

```
fac5 = 1; For[n = 1, n ≤ 5, n++, fac5 *= n]
fac5
```

```
120
```

Advanced: `Do` is a dynamic “Block” scoping construct, so although it localizes its iterator variable, it does so dynamically. Consider the following example from Shifrin. (But please do not use such constructs!)

```
Clear[a, i]
a := i * i
Do[Print[a], {i, 5}]
```

```
1
4
9
16
25
```

Indexed Variables

Unassigned Indexed Variables

Sometimes we need to create many names variables. For example, we may need to construct a large equation system with variables that are names but not assigned values. We can use indexed variables for this. Here is a small example.

```
ClearAll[x]
Solve[{x[1] + 3 x[2] == 7 && 3 x[1] - 2 x[2] == -1}, {x[1], x[2]}]
```

```
{{x[1] → 1, x[2] → 2}}
```

If we need many such variables we can easily construct them.

```
ClearAll[x]
vars = Array[x, 10]
```

```
{x[1], x[2], x[3], x[4], x[5], x[6], x[7], x[8], x[9], x[10]}
```

When solving nonlinear systems (e.g., with `FindRoot`), we may need to associate initial values with these variables, leaving the variables with no assigned values.

```
init = Map[{#, 0} &, vars]
```

```
{{x[1], 0}, {x[2], 0}, {x[3], 0}, {x[4], 0}, {x[5], 0},
 {x[6], 0}, {x[7], 0}, {x[8], 0}, {x[9], 0}, {x[10], 0}}
```

```
mA = RandomInteger[100, {5, 5}]
x = Range[5]
b = mA.x
Clear[x]
xvars = Array[x, 5]
Solve[mA.xvars == b, xvars]
DownValues[xvars]
```

```
{{21, 75, 21, 69, 39}, {56, 18, 25, 88, 2},
 {54, 67, 28, 26, 52}, {86, 15, 1, 68, 24}, {78, 13, 87, 87, 16}}
```

```
{1, 2, 3, 4, 5}
```

```
{705, 529, 636, 511, 793}
```

```
{x[1], x[2], x[3], x[4], x[5]}
```

```
{{x[1] → 1, x[2] → 2, x[3] → 3, x[4] → 4, x[5] → 5}}
```

```
{}
```

DownValues for Associative Arrays

An indexed variable can be used to create an associative array: we can use an integer as an index, and we assign a value at each index.

```
Clear[x]
DownValues[x]  (* x has no DownValues *)
x[1] = 0  (* create a DownValue for x *)
DownValues[x]  (* x is now an indexed variable; its DownValues is not empty *)
```

```
{}
```

```
0
```

```
{HoldPattern[x[1]] :↦ 0}
```

Note that because `Set` has the attribute `HoldFirst`, we have to `Evaluate` vars before we can make the assignment to the indexed variables. (Otherwise, we would just rebind the name `vars` to a new value: the array of zeros.)

```
Clear[x, xvars]
xvars = Array[x, 5]
Evaluate[xvars] = ConstantArray[0, 5]
DownValues[x]
```

```
{x[1], x[2], x[3], x[4], x[5]}
```

```
{0, 0, 0, 0, 0}
```

```
{HoldPattern[x[1]] :↦ 0, HoldPattern[x[2]] :↦ 0,
 HoldPattern[x[3]] :↦ 0, HoldPattern[x[4]] :↦ 0, HoldPattern[x[5]] :↦ 0}
```

One may also use the `Scan` command to do simple initialization.

```
Clear[x]
data = Range[5];
Scan[(x[#] = 0) &, Range[5]] (* note the parentheses *)
DownValues[x]
```

```
{HoldPattern[x[1]] :↦ 0, HoldPattern[x[2]] :↦ 0,
 HoldPattern[x[3]] :↦ 0, HoldPattern[x[4]] :↦ 0, HoldPattern[x[5]] :↦ 0}
```

In principle, any expression can be used as an index. In practice, the use of symbols is likely to create confusion, because indexes are evaluated. Here is a simple example, demonstrating the the results are generally not what we intended.

```
Clear[f, x]
f[x] = x * x;  (* use the symbol `x` as an index *)
DownValues[f] (* we see we need a literal `f[x]` *)
f[x]  (* literal `f[x]` works as expected *)
f[y]  (* `f[y]` is unrecognized, so evaluates to self *)
x = 2  (* now `x` will evaluate to 2 *)
f[x]  (* `f[2]` is unrecognized! *)
```

```
{HoldPattern[f[x]] :↦ x^2}
```

```
x^2
```

```
f[y]
```

```
2
```

```
f[2]
```

In contrast, the use of patterns as indexes can be a very good idea.

DownValues for Function Definition

We can use any pattern to index a variable. If we name the pattern and use delayed assignment (`SetDelayed`), we can use that name on the right. This gives us another and very useful way to define

functions.

```
ClearAll[f, x]
x = 5  (* give `x` a global value *)
f[x_] := x * x  (* `x` on the right matches the pattern on the left! *)
DownValues[f]  (* we have defined a transformation rule *)
f[x]  (* remember, the global `x` evaluates to 5 *)
f[y]  (* the symbol `y` will be transformed symbolically *)
```

```
5
```

```
{HoldPattern[f[x_]] :→ x x}
```

```
25
```

```
y^2
```

Note that when you define functions using pattern matching, that reuse of a name for the function does not remove the old definition if your new definition uses a new pattern.

```
Clear[f, x, y]
f[x_] := x * x
f[x_, y_] := x * x + y * y
DownValues[f]
f[2]  (* uses first definition *)
f[2, 3] (* uses second definition *)
```

```
{HoldPattern[f[x_]] :→ x x, HoldPattern[f[x_, y_]] :→ x x + y y}
```

```
4
```

```
13
```

We can define our functions with any patterns, which allows conditional definitions. For example, we might want to allow only non-negative inputs to a production function.

```
Clear[f, x]
f[n_ /; n ≥ 0, k_ /; k ≥ 0] := n^0.6 k^0.4
f[1, 1]
f[-1, -1]  (* not defined *)
```

```
1.
```

```
f[-1, -1]
```

As a more useful example, consider defining a function $\sin(x)/x$, which is undefined at 0. We can "plug" this discontinuity by using conditional matching. Note that even though we define the specific case `f[0]` after defining the general case, the `DownValues` are re-ordered so that the specific case is tried first.