

# CT561: Systems Modelling and Simulation

## Week 6: Simulation with Vensim & Introduction to R

<https://github.com/JimDuggan/CT561>

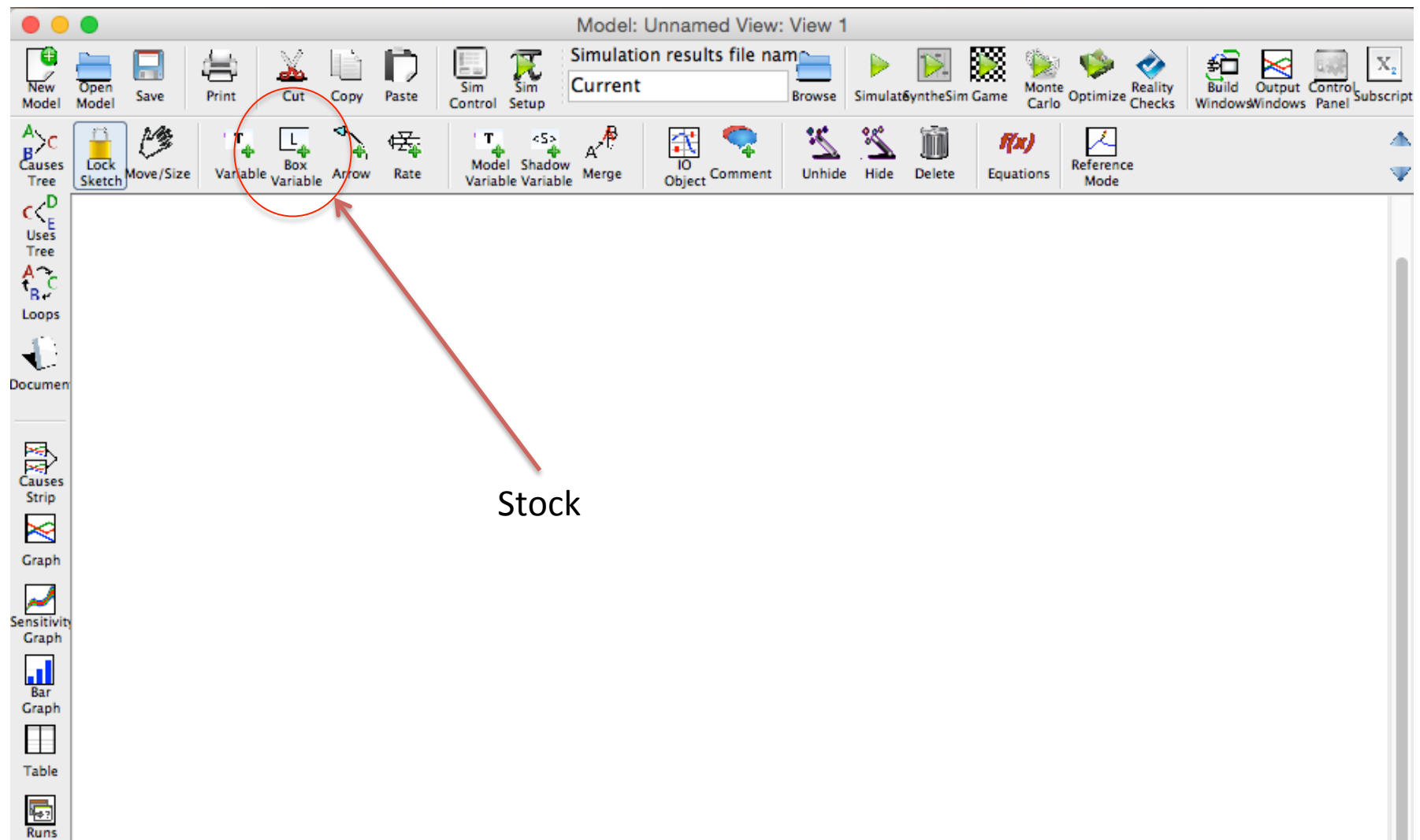
Dr. Jim Duggan,  
Information Technology,  
School of Engineering & Informatics



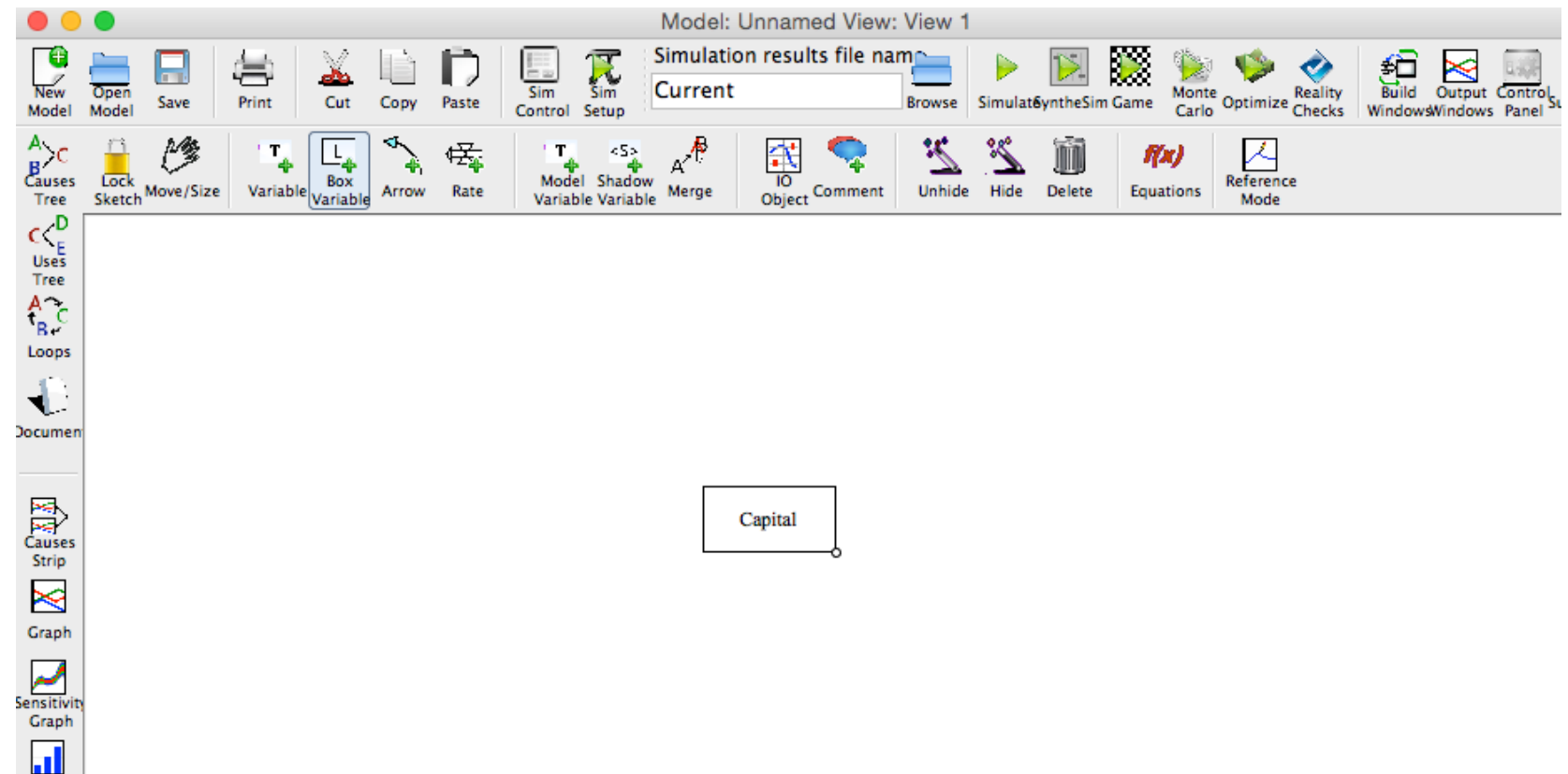
# Overview

- Vensim PLE
  - Supports model building with stocks and flows
  - Equation editor
  - Time specs (Start, Finish, DT)
  - Supports graphing
- R and deSolve
  - Functional programming language
  - Supports data science methodology
  - Calibration and sensitivity analysis

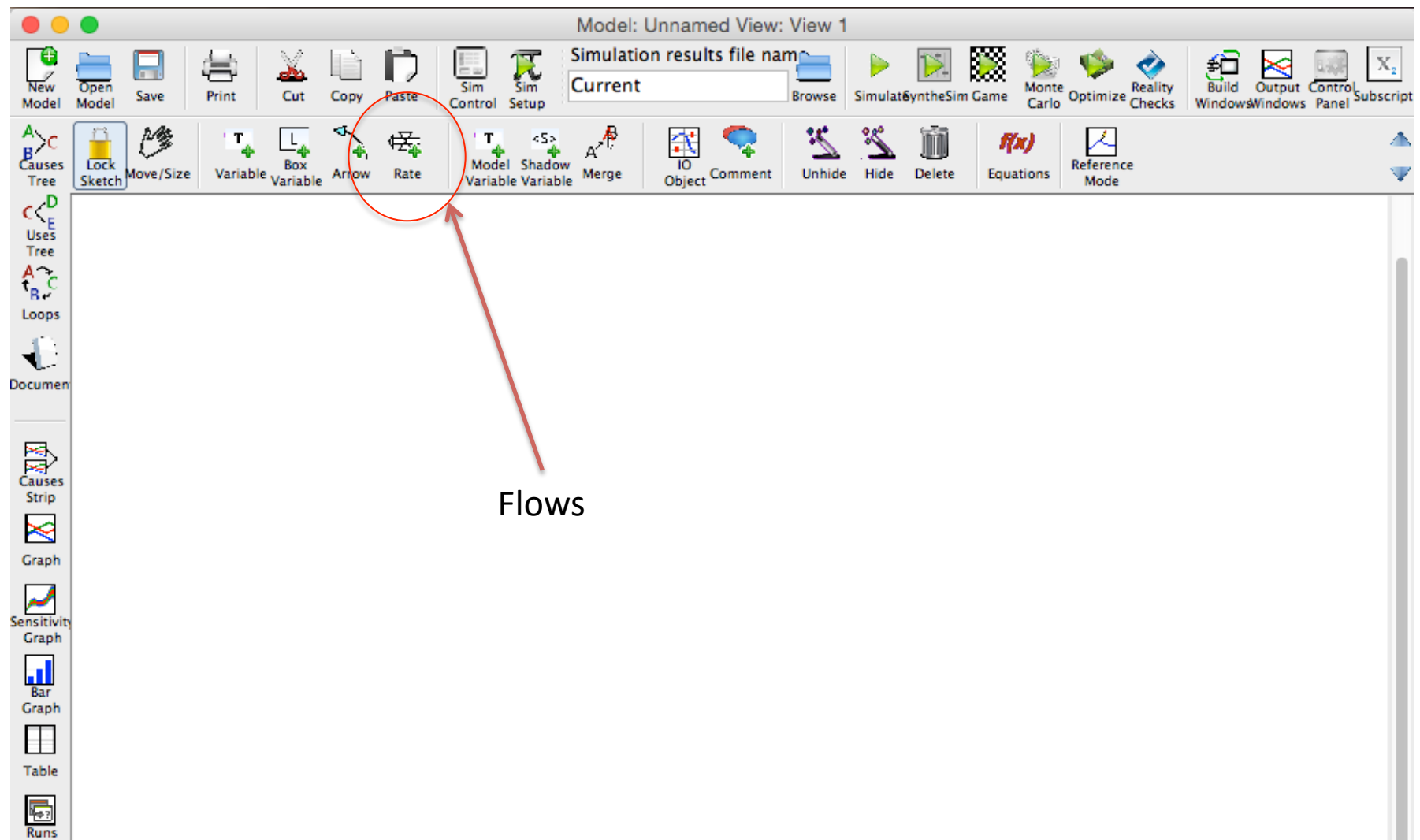
# Vensim: Adding a Stock



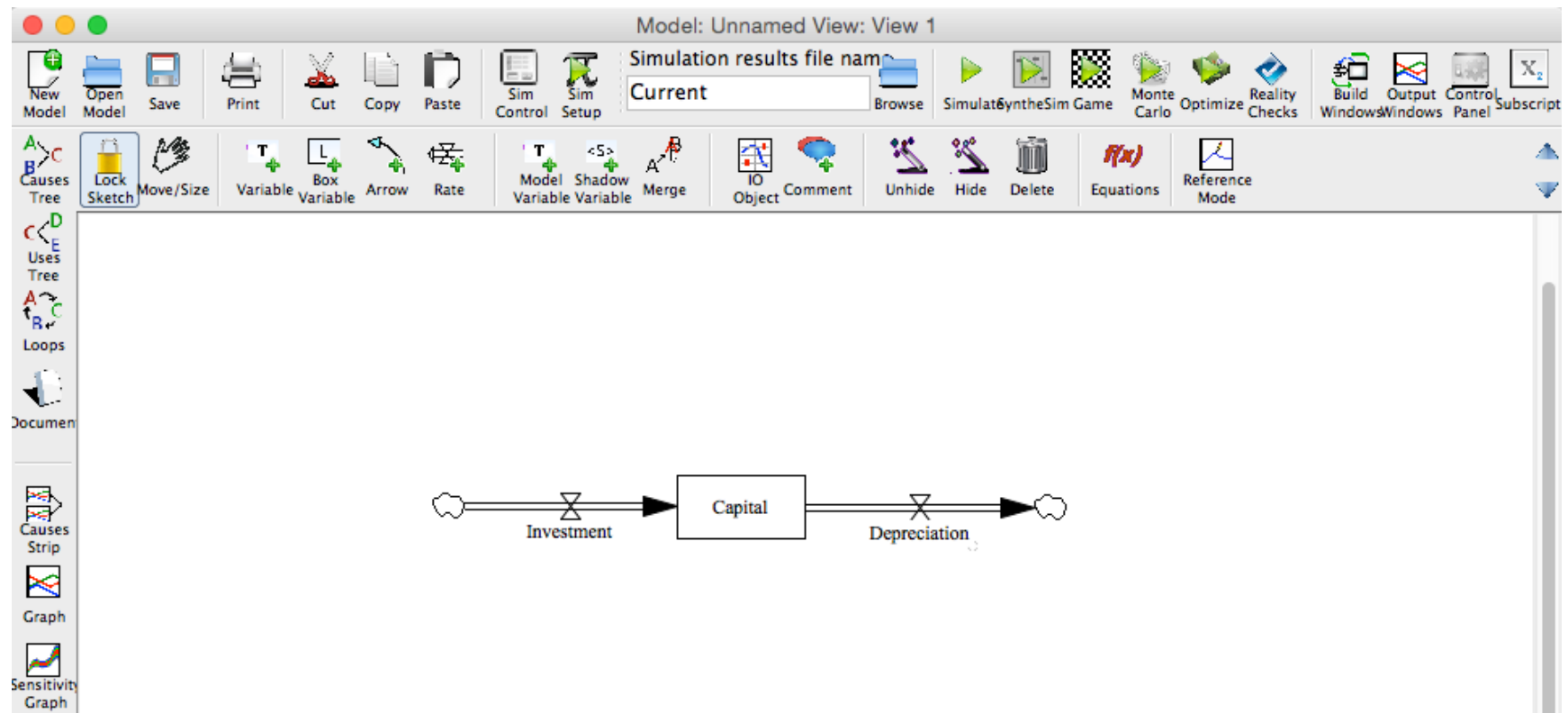
# Capital Stock



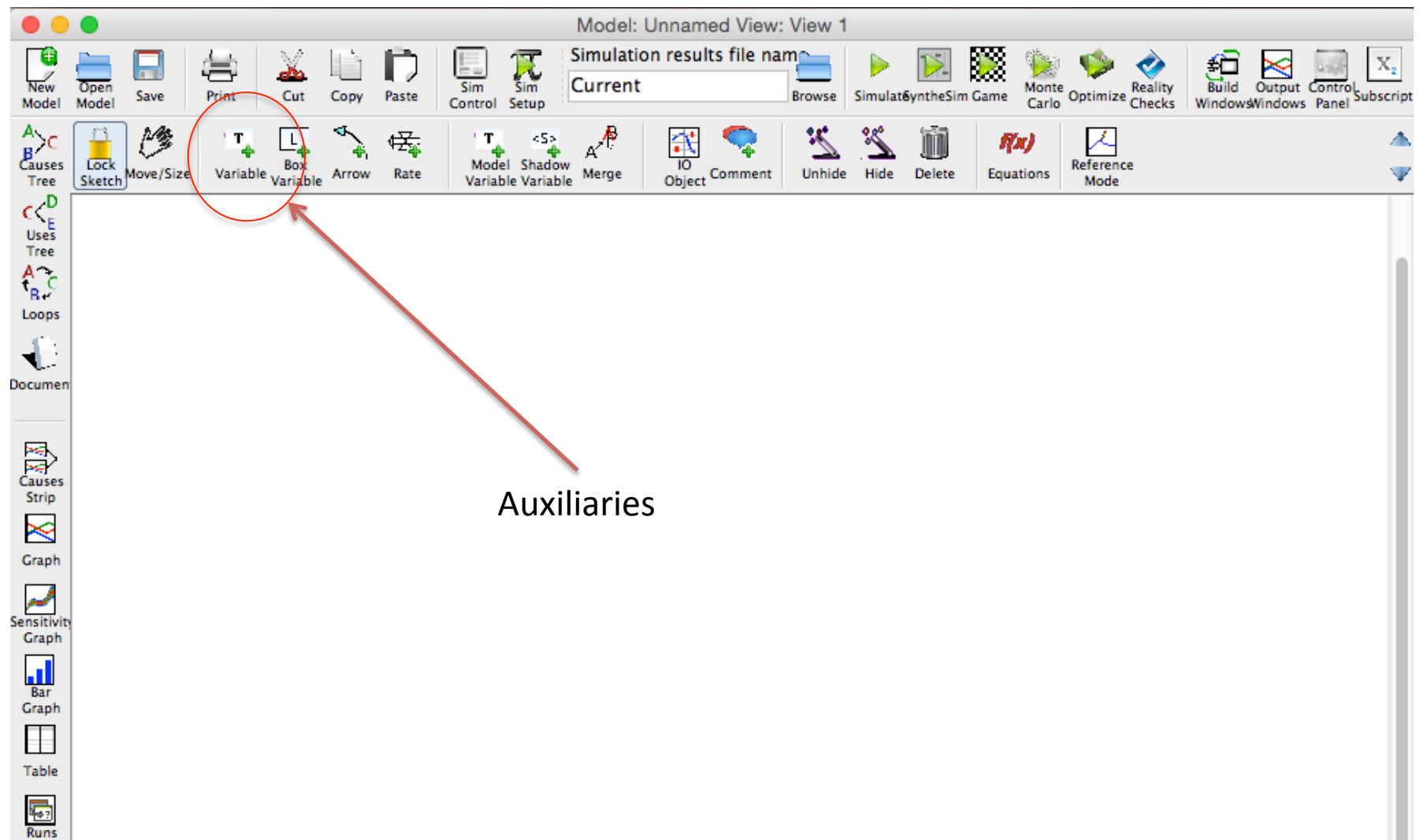
# Vensim: Adding Flows



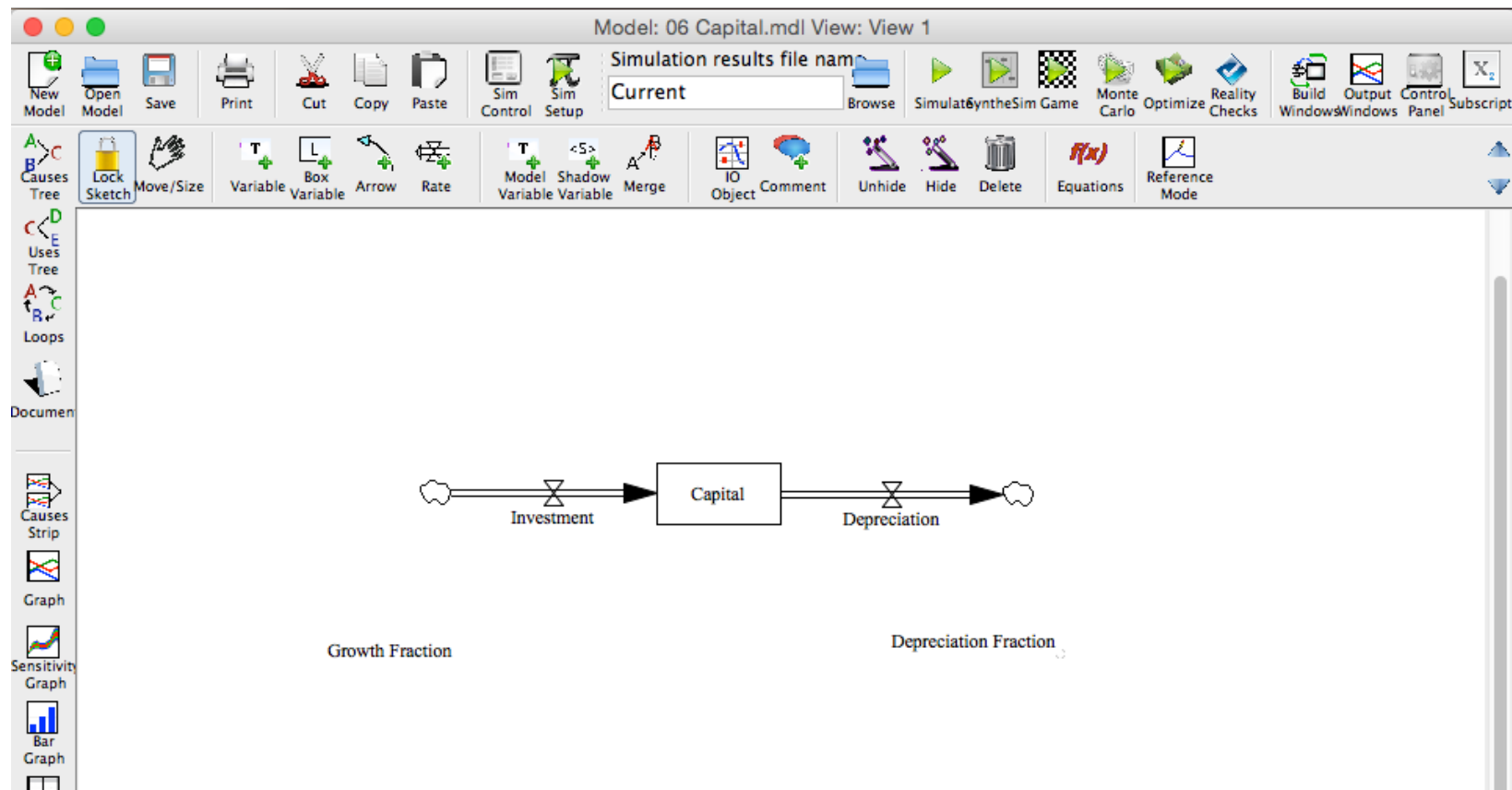
# Investment and Depreciation Flows



# Vensim: Adding Auxiliaries

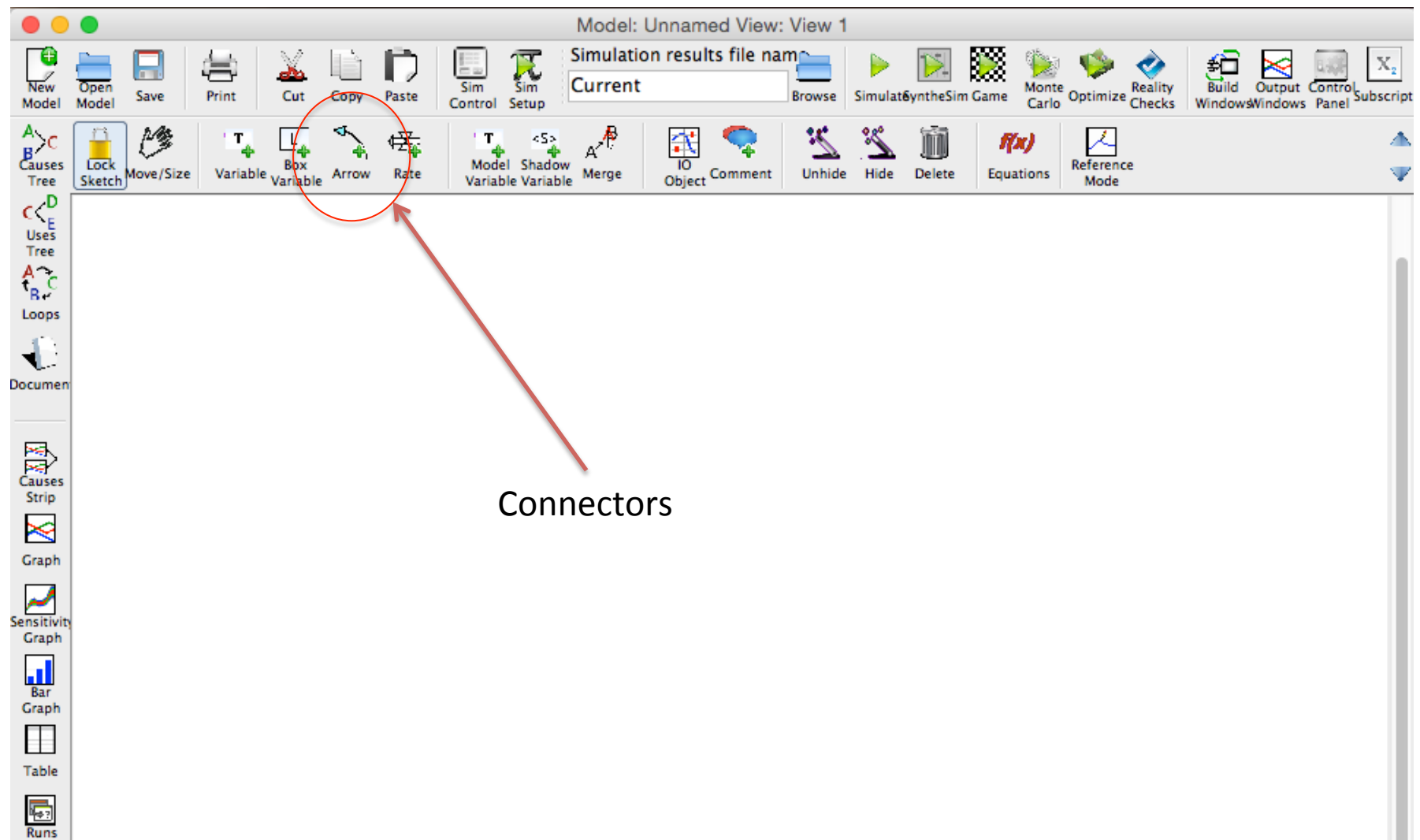


# Growth and Depreciation Fractions

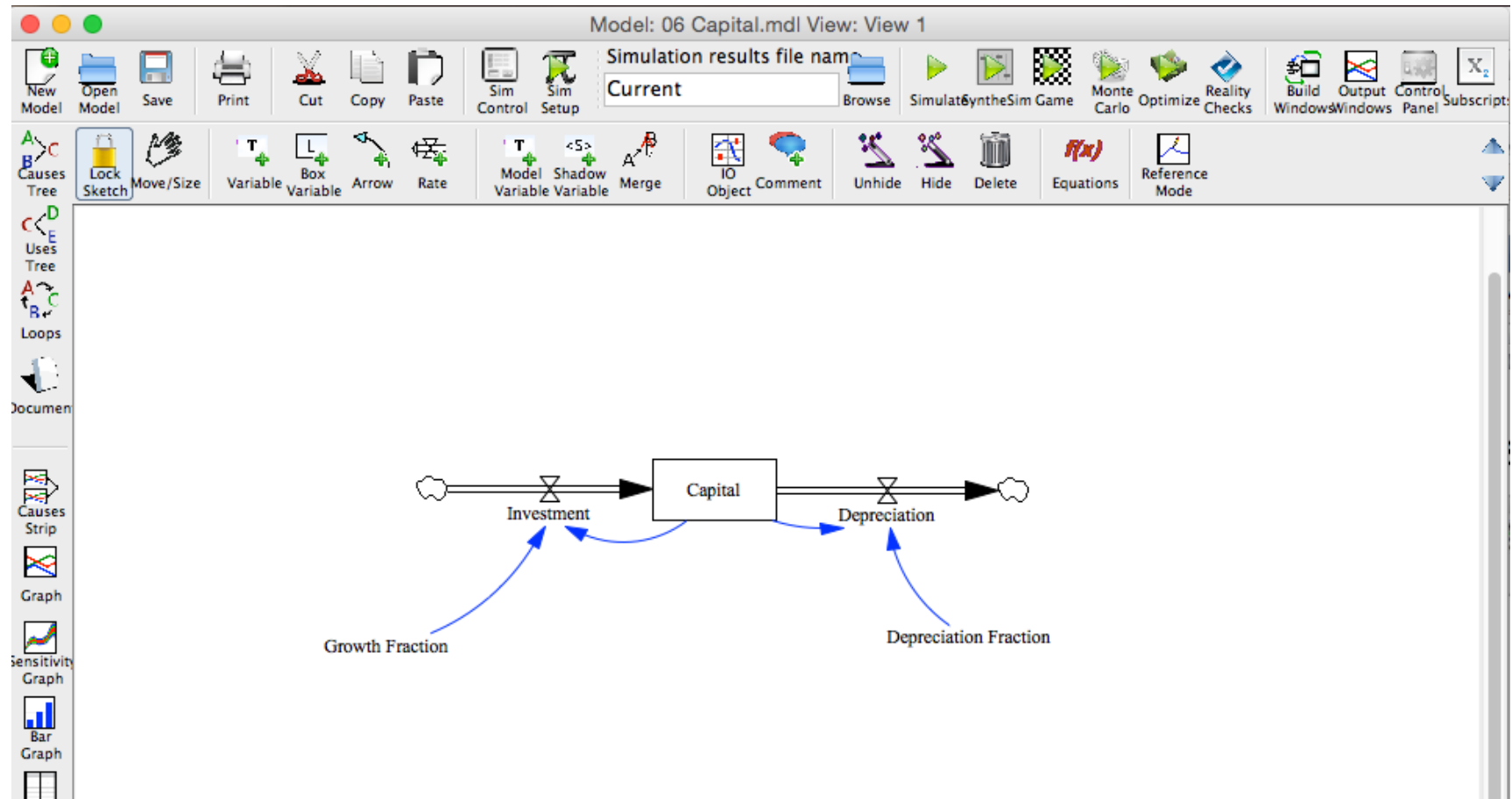




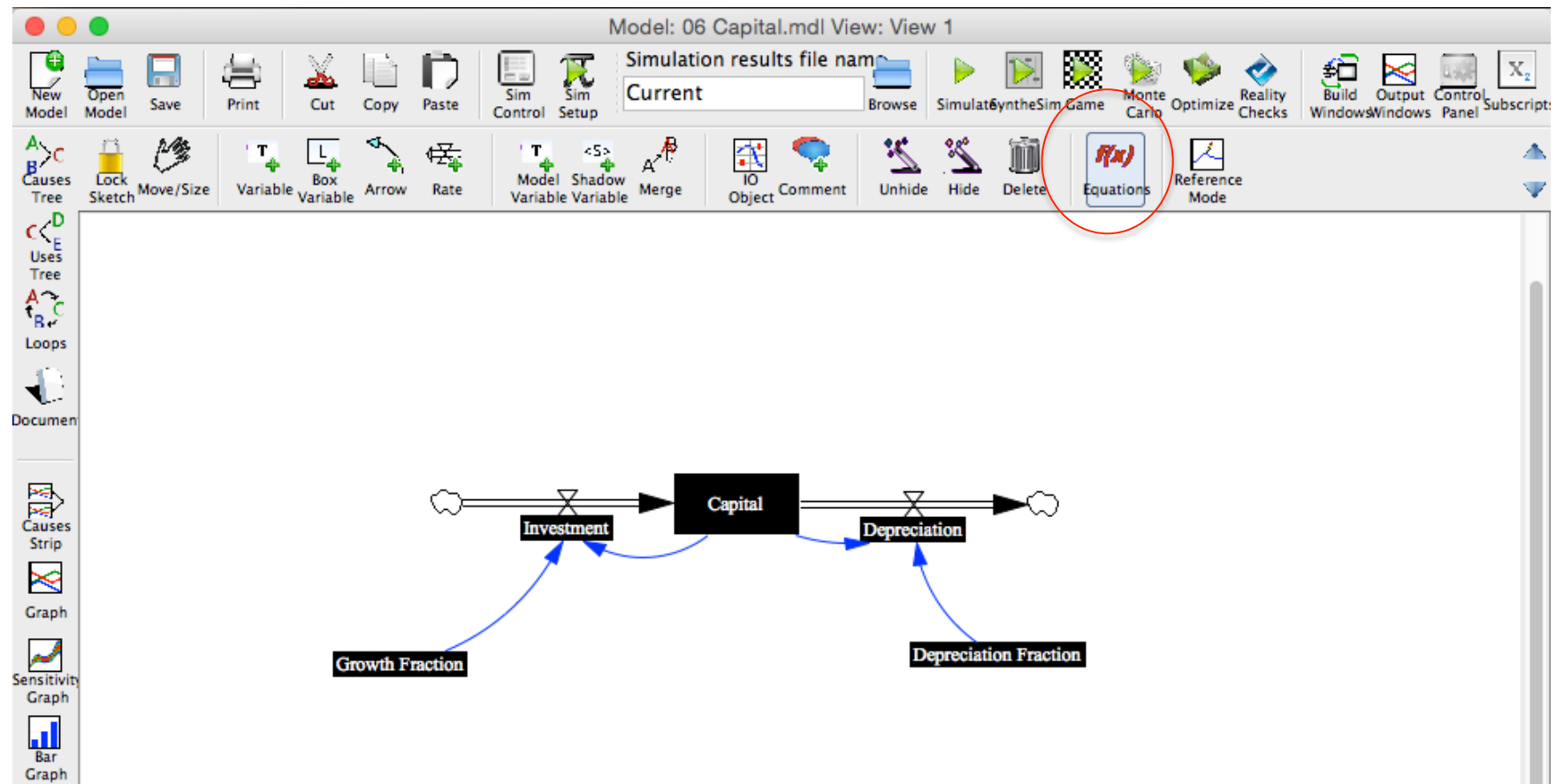
# Vensim: Adding Connectors



# Connect from cause to effect variables



# Vensim: Edit equations



# Adding the Capital Equation

Variable Information

Name:

Type:  Sub-Type:

Units:   ☐ Supplementary

Group:  Min:  Max:

Equations

Subscripts:

= INTEG (

Initial Value:

Functions:

Keypad Buttons

Subscripts:

ABS  
DELAY FIXED  
DELAY1  
DELAY1I  
DELAY3  
DELAY3I  
EXP  
GET 123 CONSTANTS

7 8 9 + :AND:  
4 5 6 - :OR:  
1 2 3 \* :NOT:  
0 E . / :NA:  
( ) , ^ <>  
> >= = < <=  
[ ] ! { }  
Undo -> {( ) }

# Adding the fractions

Variable Information

Name: Growth Fraction

Type: Constant Sub-Type: Normal

Units: Check

Group: .06 Capital Min

Equations

Subscripts: 0.05

OK Chk

Variable Information

Name: Depreciation Fraction

Type: Constant Sub-Type: Normal

Units: Check

Group: .06 Capital Min

Equations

Subscripts: 0.02

OK Chk

# Adding the flows

Variable Information

Name: Investment

Type: Auxiliary Sub-Type: Normal

Units:

Group: .06 Capital Min

Equations

Subscripts

=

Capital\*Growth Fraction

OK Chk

Variable Information

Name: Depreciation

Type: Auxiliary Sub-Type: Normal

Units:

Group: .06 Capital Min

Equations

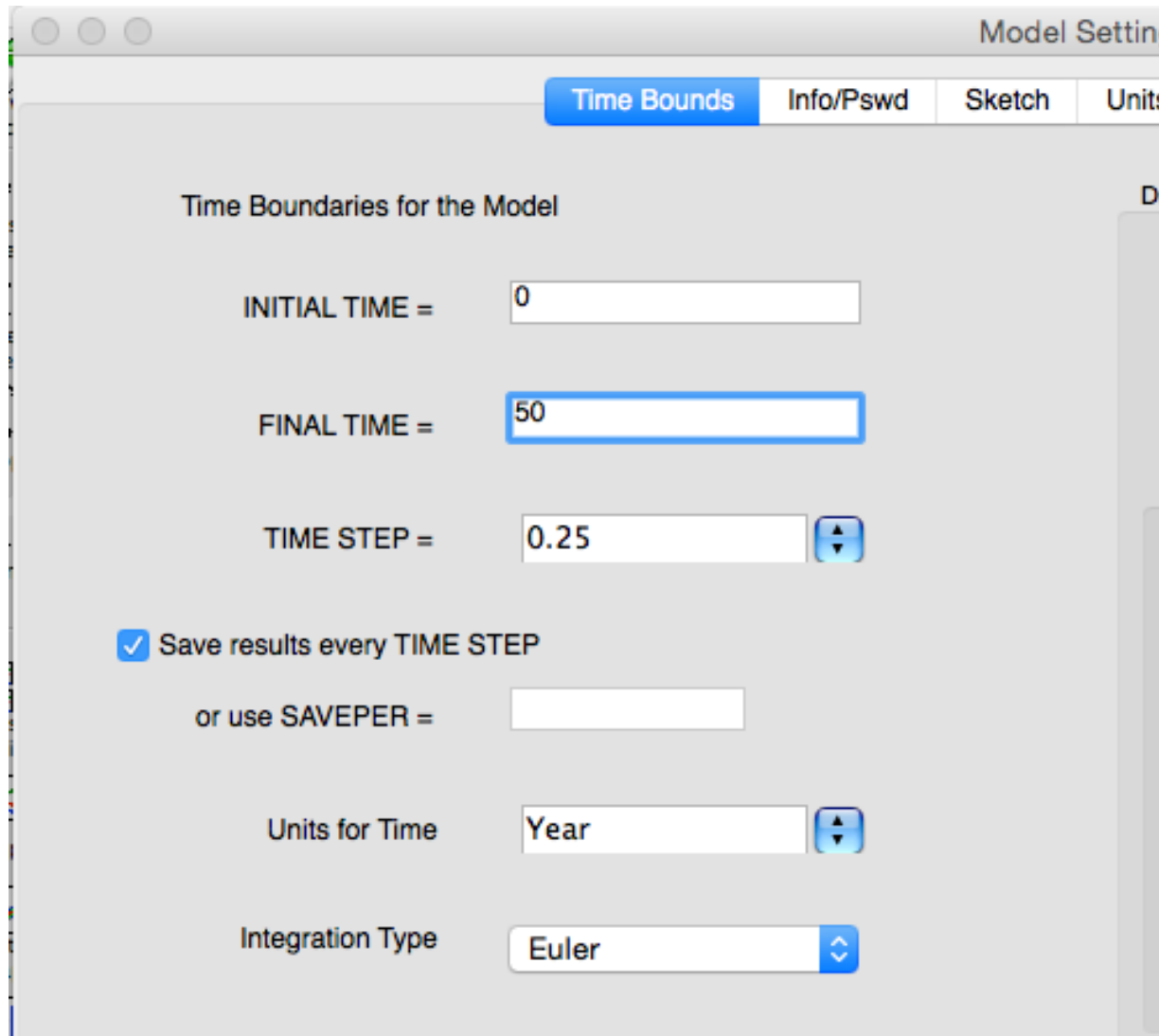
Subscripts

=

Capital\*Depreciation Fraction

OK Chk

# Setting the run parameters



The screenshot shows a software window titled "Model Settings" with a tabbed interface. The "Time Bounds" tab is selected, showing fields for "INITIAL TIME = 0", "FINAL TIME = 50", and "TIME STEP = 0.25". There is a checkbox for "Save results every TIME STEP" which is checked, and an empty field for "or use SAVEPER =". Below these are dropdown menus for "Units for Time" (set to "Year") and "Integration Type" (set to "Euler").

Model Settings

Time Bounds Info/Pswd Sketch Units

Time Boundaries for the Model

INITIAL TIME = 0

FINAL TIME = 50

TIME STEP = 0.25

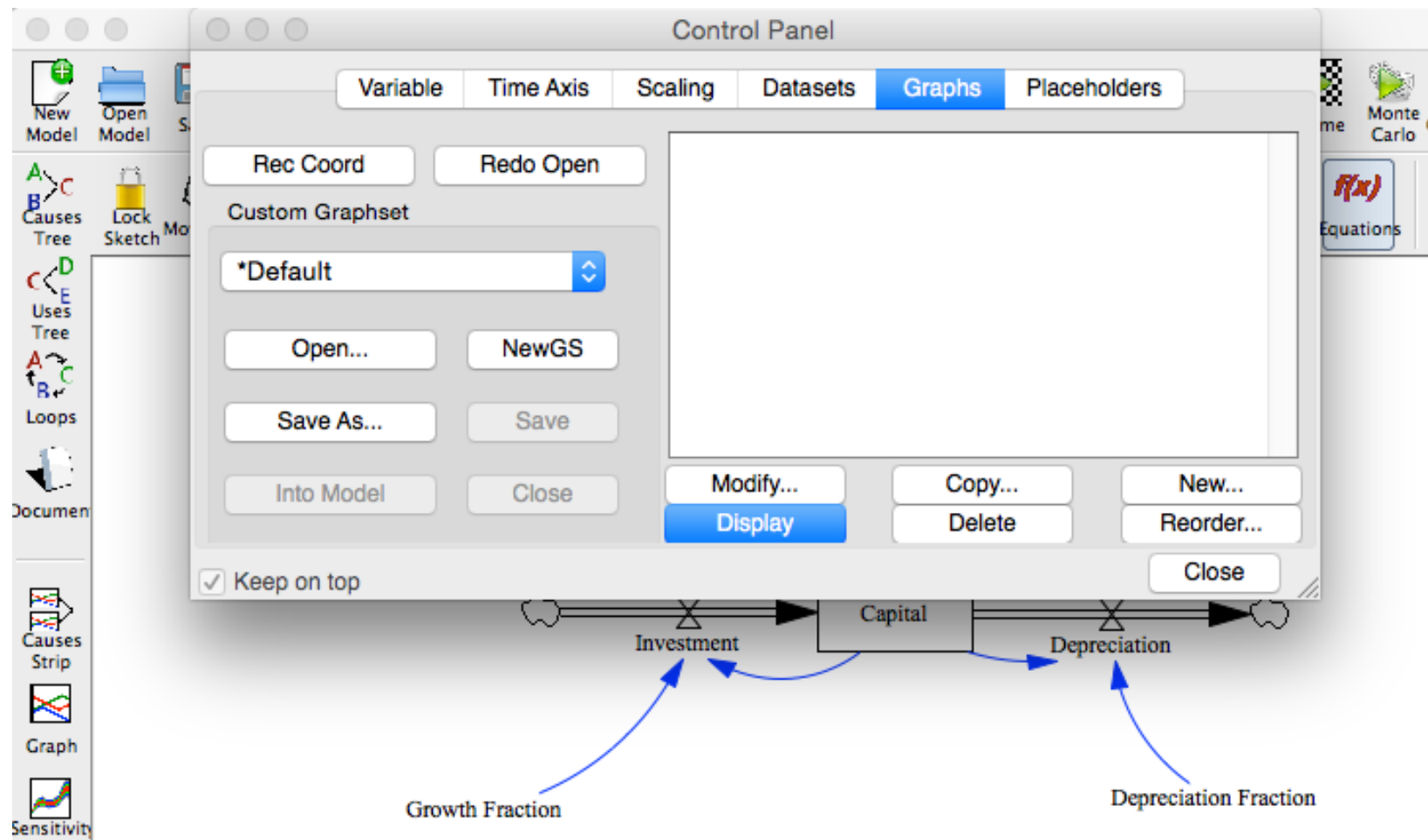
☒ Save results every TIME STEP

or use SAVEPER =

Units for Time Year

Integration Type Euler

# Add a graph

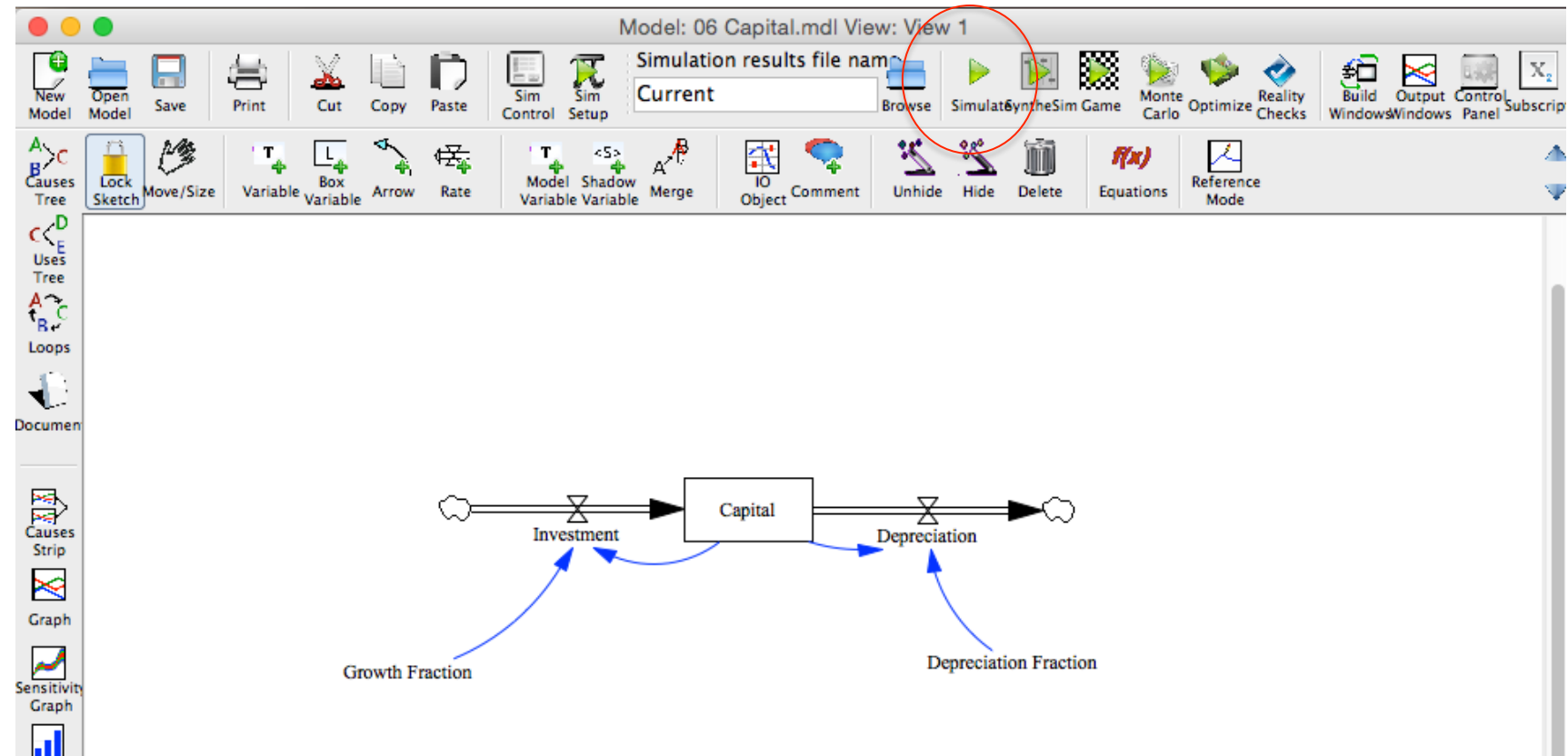




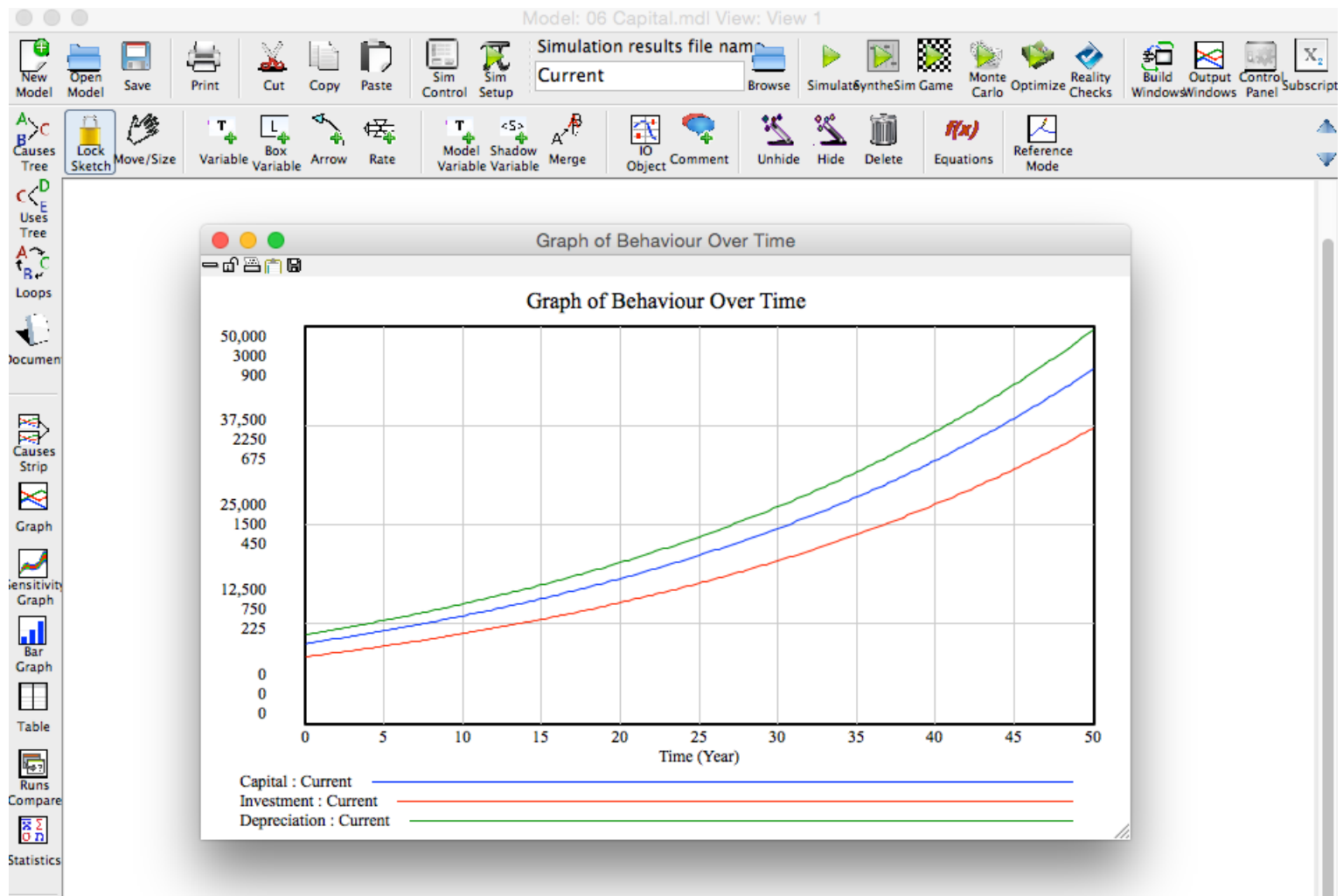
# Finishing the graph

Name	<input type="text" value="Output"/>		Hide:	<input type="checkbox"/>	Title
Title	<input type="text" value="Graph of Behaviour Over Time"/>				
X-Axis	<input type="text"/>	<input type="button" value="Sel"/>	X Label	<input type="text"/>	
X-min	<input type="text"/>	X-max	<input type="text"/>	X-divisions	<input type="text"/>
Stamp	<input type="text"/>			Comment	<input type="text"/>
Type	<input checked="" type="radio"/> Norm <input type="radio"/> Cum <input type="radio"/> Stack <input type="checkbox"/> Dots <input type="checkbox"/> Fill				
Scale	Variable		Dataset	Label	
<input type="checkbox"/>	<input type="text" value="Capital"/>	<input type="button" value="Sel"/>	<input type="text"/>	<input type="text"/>	
<input type="checkbox"/>	<input type="text" value="Investment"/>	<input type="button" value="Sel"/>	<input type="text"/>	<input type="text"/>	
<input type="checkbox"/>	<input type="text" value="Depreciation"/>	<input type="button" value="Sel"/>	<input type="text"/>	<input type="text"/>	
<input type="checkbox"/>	<input type="text"/>	<input type="button" value="Sel"/>	<input type="text"/>	<input type="text"/>	

# Running the simulation



# Display the results



# Introduction to R

# Overview

- R's *mission* is to enable the best and most thorough exploration of data possible (Chambers 2008).
- It is a dialect of the S language, developed at Bell Laboratories
- ACM noted that S “*will forever alter the way people analyze, visualize, and manipulate data*”

---

## ACM HONORS DR. JOHN M. CHAMBERS OF BELL LABS WITH THE 1998 ACM SOFTWARE SYSTEM AWARD FOR CREATING "S SYSTEM" SOFTWARE

New York, March 23, 1999...The Association for Computing Machinery (ACM) today named Dr. John M. Chambers of Bell Labs as the recipient of the 1998 Software System Award for developing the S System, an innovative software program that helps users to manage and extract useful information from data.

The ACM's citation notes that Dr. Chambers' work "will forever alter the way people analyze, visualize, and manipulate data . . . S is an elegant, widely accepted, and enduring software system, with conceptual integrity, thanks to the insight, taste, and effort of John Chambers."

The System Software Award recognizes those who develop software systems having a lasting influence. It will be presented on May 15, 1999 during a special ACM awards banquet in New York City, and will be accompanied by a \$10,000 prize. Financial support is provided by IBM.

### About The "S System"

The first versions of S in the 1970s pioneered the use of data visualization and interactive statistical computing. Subsequent versions provided richly enhanced modeling capability, and user extensibility, based on its functional object-based approach.

Still more recent versions provide a powerful class/method structure, new techniques to deal with large objects, extended interfaces to other languages and files, object-based documentation compatible with HTML, and powerful interactive programming techniques. The commercial version, S-Plus, is used across many disciplines where analysts must struggle with creative ways to manage and extract useful information from data. More information about S is available at <http://cm.bell-labs.com/stat/S>.

<http://www.acm.org/announcements/ss99.html>

# R Studio: A Data Science Workbench

R Code

Environment/State

The screenshot shows the R Studio interface with four main panels. A red border highlights the entire interface. Blue dashed arrows point from the labels to specific parts of the interface:

- R Code:** Points to the script editor showing R code for sampling and creating a table.
- Environment/State:** Points to the Environment pane showing the current state of the workspace.
- Interactive console:** Points to the console showing the output of the executed R code.
- File System:** Points to the Files pane showing the project's file structure.

**Script Editor (01 Introduction.R):**

```
83 s4<-sample(1:3,20,replace=TRUE)
84 s2=s4
85
86 # Gather data on the frequency of (whole) numbers in a vector
87 t1<-table(s1)
88 props<-prop.table(t1)
89
90
91
92
93
94
95
96
90:1 (Top Level) R Script
```

**Environment (Global Environment):**

Values	
b1	logi [1:2] FALSE TRUE
b2	logi [1:5] FALSE TRUE FALSE TRUE FALSE
c1	chr [1:5] "Odd" "Even" "Odd" "Even" "Odd"
ind	2L
index	5L
props	table [1:3(1d)] 0.4 0.3 0.3
r	24
s	num [1:101] 51.4 55 57.2 57.3 58.6 ...
s1	int [1:20] 2 1 1 3 1 3 2 1 1 1 ...
s2	int [1:20] 2 3 2 2 2 2 1 2 2 1 ...

**Console:**

```
> s2=s4
[1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
> t1<-table(s1)
> props<-prop.table(t1)
> props
s1
 1  2  3
0.4 0.3 0.3
>
>
```

**Files:**

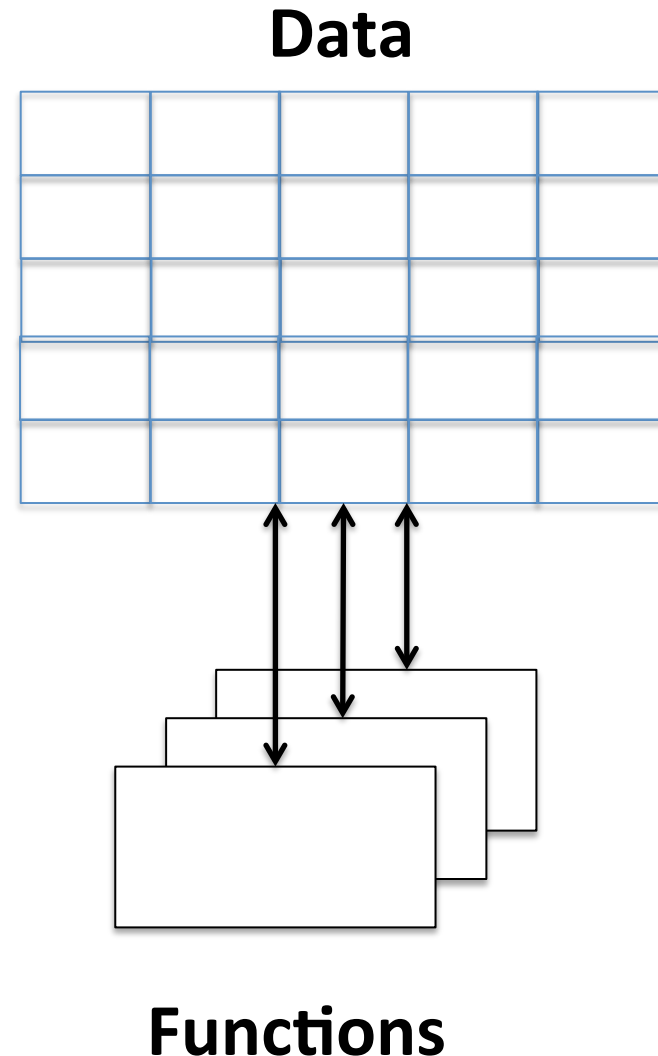
Name	Size	Modified
..		
01 Introduction.R	1.9 KB	Sep 3, 2015, 2:23 PM
01 Vectors.pdf	892 KB	Sep 3, 2015, 2:26 PM

Interactive console

File System

# Functional Programming

- R is a functional programming language, where software programs are organized into *functions* that can be called to transform data.
- Users of R should adopt the habit of creating simple functions which will make their work **more effective** and also **more trustworthy** (Chambers 2008).



# Data Structures - Vector

- The fundamental data type in R is the vector
- Similar to 1-D arrays in C and Java
- A variable that contains a sequence of elements that have the same data type (Matloff 2009).
- Create using `c(e1, ..., en)`
- **Assignment statement** `<-`

v1

1
4
9
16
25

```
> v1<-c(1,4,9,16,25)
> v1
[1] 1 4 9 16 25
```



# Index

- The concept of an index is powerful in R, as it allows access to individual data elements of a vector, using the square brackets notation.
- In R, unlike programming languages such as C and Java, the index for a vector starts at 1.

```
> v1  
[1] 1 4 9 16 25  
  
> v1[1]  
[1] 1  
  
> v1[2]  
[1] 4  
  
> v1[5]  
[1] 25
```

# Creating sequences

- The colon operator (:) generates regular sequences within a specified range.

```
> v1<-1:10  
> v1  
[1] 1 2 3 4 5 6 7 8 9 10  
  
> v2<-3:13  
> v2  
[1] 3 4 5 6 7 8 9 10 11 12 13
```

# Using **seq()** function

- Useful function to create sequences, and allows for further detail via the **by** argument

```
> v6<-seq(1,5)
> v6
[1] 1 2 3 4 5

> v7<-seq(1,5,by=.5)
> v7
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0

> v8<-seq(1,100,by=20)
> v8
[1] 1 21 41 61 81
```

# Sequences as indices

- Sequences can be used as indices for vectors, with the minus sign used for exclusion

```
> v1  
[1] 1 4 9 16 25
```

```
> v1[1:2]  
[1] 1 4
```

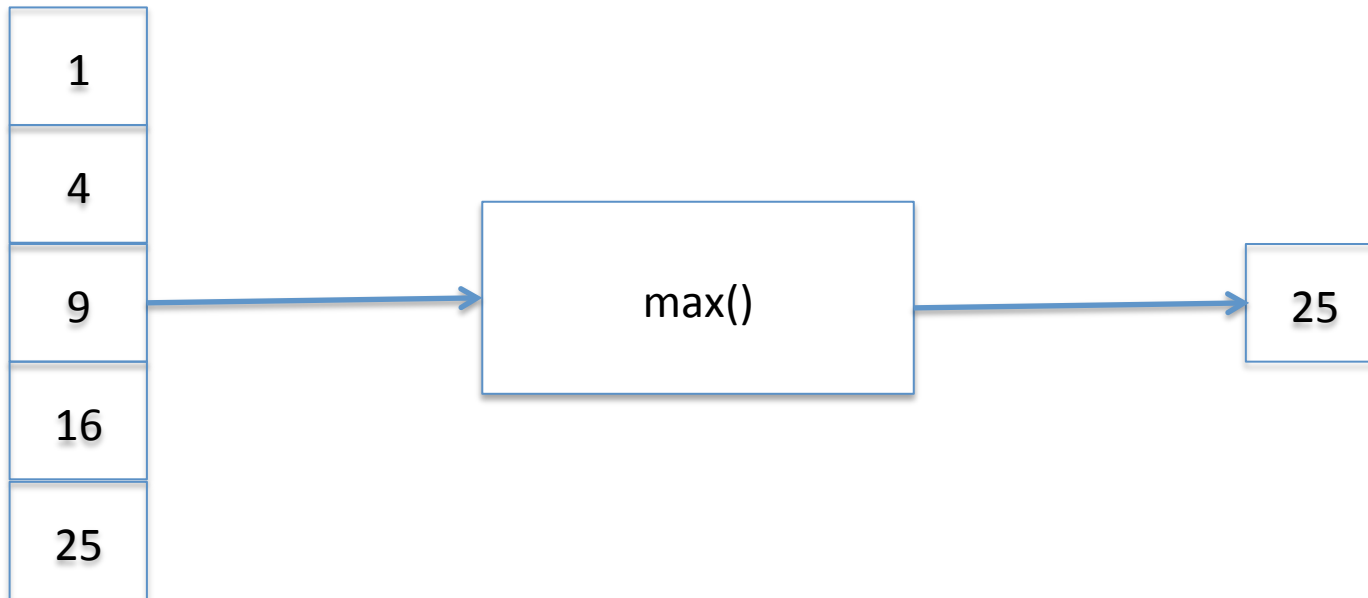
```
> v1[-1]  
[1] 4 9 16 25
```

```
> v1[-(1:3)]  
[1] 16 25
```

# max function

Input Vector

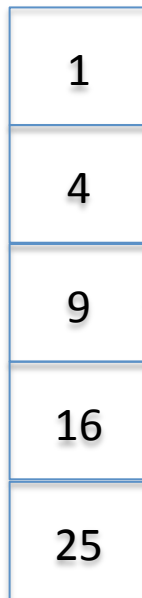
Output Vector



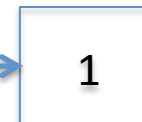
```
> v1  
[1] 1 4 9 16 25  
> max(v1)  
[1] 25
```

# min function

Input Vector



Output Vector



min()

```
> v1  
[1] 1 4 9 16 25  
> min(v1)  
[1] 1
```

# Challenge 1.1

- Create an R vector of squares of 1 to 10
- Find the minimum
- Find the maximum

# Vectorization

- A powerful feature of R is that it supports *vectorization*
- Functions can operate on every element of a vector, and return the results of each individual operation in a new vector.

```
> v1
[1] 1 2 3 4 5

> r<-sqrt(v1)

> r
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
```



# Key Idea

**Input Vector**

1
4
9
16
25

**Output Vector**

1
2
3
4
5

**sqrt()**

```
graph LR; Input[Input Vector] --> sqrt[sqrt()]; sqrt --> Output[Output Vector]
```

The diagram illustrates a data transformation process. An input vector containing the values [1, 4, 9, 16, 25] is processed by a function labeled 'sqrt()'. The resulting output vector contains the values [1, 2, 3, 4, 5], which are the square roots of the input values. Arrows indicate the flow of data from the input vector to the function and then to the output vector.

# Conditional Expressions on Vectors

- Conditional expressions can be applied to vectors, and this operation returns a boolean vector

```
> v1  
[1] 1 4 9 16 25
```

```
> v1 %% 2==0  
[1] FALSE TRUE FALSE TRUE FALSE
```

# Boolean vectors used as indices

- A target vector can be filtered by the TRUE locations in a boolean vector.

```
> v1
[1] 1 4 9 16 25

> b1<-v1 %% 2 == 1

> b1
[1] TRUE FALSE TRUE FALSE TRUE

> v1[b1]
[1] 1 9 25
```

# which()

- Give the TRUE indices of a logical object, allowing for array indices.

```
> v1  
[1] 1 4 9 16 25  
> ind<-which(v1==4)  
> ind  
[1] 2  
> v1[ind]  
[1] 4
```

# Naming vector elements

- The elements of a vector can also be given names, and this can be useful in defining parameters for further analysis. The name can be used as an index.

```
> names(v1)<-  
c("Var1","Var2","Var3","Var4","Var5")  
  
> v1  
Var1 Var2 Var3 Var4 Var5  
  1   4   9  16  25  
  
> v1["Var5"]  
Var5  
 25
```

# Vectorized if/else

- Vectors can also be processed using the vectorized **ifelse(b,u,v)** function, which accepts a boolean vector **b** and allocates the element-wise results to be either **u** or **v**.

```
> v1
[1] 1 4 9 16 25

> c1<-ifelse(v1%%2==0,"Even","Odd")

> c1
[1] "Odd" "Even" "Odd" "Even" "Odd"
```

# Challenge 1.2

- Write 2 “parallel” vectors that store a course code in one, and the number of students in the other. Write a script that returns the number of students (output) in a course (input). Make use of the == operator for string comparison.

```
> "CT102"=="CT101"  
[1] FALSE
```

```
> "CT102"=="CT102"  
[1] TRUE
```

# Summary

- Vectors: a key data structure in R
- Vectorization – apply an operation on all elements and return a new vector
- Boolean vectors – very useful!



# References

- Chambers, J. 2008. Software for data analysis: programming with R. Springer Science & Business Media. Chicago.
- Chang, W. 2013. R Graphics Cookbook. "O'Reilly Media, Inc." Sebastapol, CA.
- Matloff, N. 2009. The Art of R Programming. No Starch Press, San Francisco, CA.