4 Pandas 2: Plotting

Lab Objective: *Clear, insightful visualizations are a crucial part of data analysis. To facilitate quick data visualization, pandas includes several tools that wrap around matplotlib. These tools make it easy to compare different parts of a data set, explore the data as a whole, and spot patterns and correlations the data.*

Overview of Plotting Tools

The main tool for visualization in pandas is the `plot()` method for `Series` and `DataFrames`. The method has a keyword argument `kind` that specifies the type of plot to draw. The valid options for `kind` are detailed below.

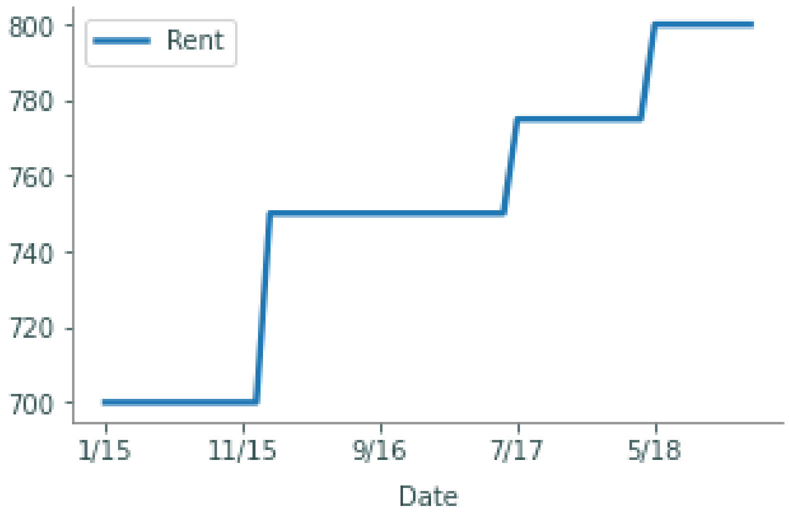
Plot Type	`plot()` ID	Uses and Advantages
Line plot	`"line"`	Show trends ordered in data; easy to compare multiple data sets
Scatter plot	`"scatter"`	Compare exactly two data sets, independent of ordering
Bar plot	`"bar"`, `"barh"`	Compare categorical or sequential data
Histogram	`"hist"`	Show frequencies of one set of values, independent of ordering
Box plot	`"box"`	Display min, median, max, and quartiles; compare data distributions
Hexbin plot	`"hexbin"`	2D histogram; reveal density of cluttered scatter plots

Table 4.1: Types of plots in pandas. The plot ID is the value of the keyword argument `kind`. That is, `df.plot(kind="scatter")` creates a scatter plot. The default `kind` is `"line"`.

The `plot()` method calls `plt.plot()`, `plt.hist()`, `plt.scatter()`, and other matplotlib plotting functions, but it also assigns axis labels, tick marks, legends, and a few other things based on the index and the data. Most calls to `plot()` specify the `kind` of plot and which `Series` to use as the `x` and `y` axes. By default, the `index` of the `Series` or `DataFrame` is used for the `x` axis.

```
>>> import pandas as pd
>>> from matplotlib import pyplot as plt

>>> budget = pd.read_csv("budget.csv", index_col="Date")
>>> budget.plot(y="Rent")  # Plot rent against the index (date).
```



In this case, the call to the `plot()` method is essentially equivalent to the following code.

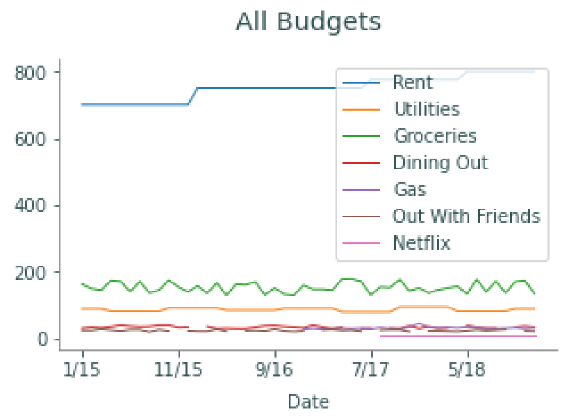
```
>>> plt.plot(budget.index, budget['Rent'], label='Rent')
>>> plt.xlabel(budget.index.name)
>>> plt.xlim(min(budget.index), max(budget.index))
>>> plt.legend(loc='best')
```

The `plot()` method also takes in many keyword arguments for matplotlib plotting and annotation functions. For example, setting `legend=False` disables the legend, providing a value for `title` sets the figure title, `grid=True` turns a grid on, and so on. For more customizations, see `https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.plot.html`.

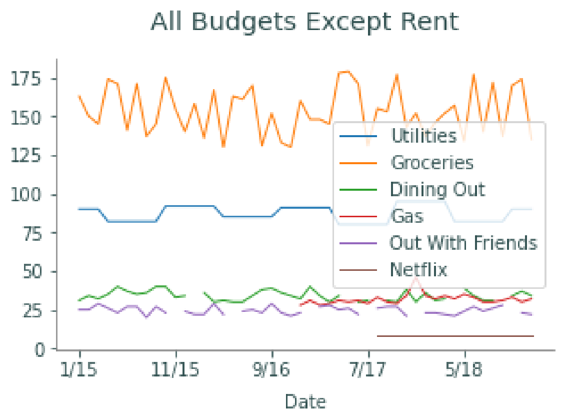
Visualizing an Entire Data Set

A good way to start analyzing an unfamiliar data set is to visualize as much of the data as possible to determine which parts are most important or interesting. For example, since the columns in a `DataFrame` share the same index, the columns can all be graphed together using the index as the x-axis. By default, the `plot()` method attempts to plot **every** `Series` (column) in a `DataFrame`. This is especially useful with sequential data, like the budget data set.

```
# Plot all columns together against the index.
>>> budget.plot(title="All Budgets",linewidth=1)
>>> budget.drop(["Rent"], axis=1).plot(linewidth=1,title="All Budgets Except ↩
    Rent")
```



(a) All columns of the budget data set on the same figure, using the index as the x-axis.



(b) All columns of the budget data set except `"Living Expenses"` and `"Rent"`.

Figure 4.1

While plotting every `Series` at once can give an overview of all the data, the resulting plot is often difficult for the reader to understand. For example, the budget data set has 9 columns, so the resulting figure, Figure 4.1a, is fairly cluttered.

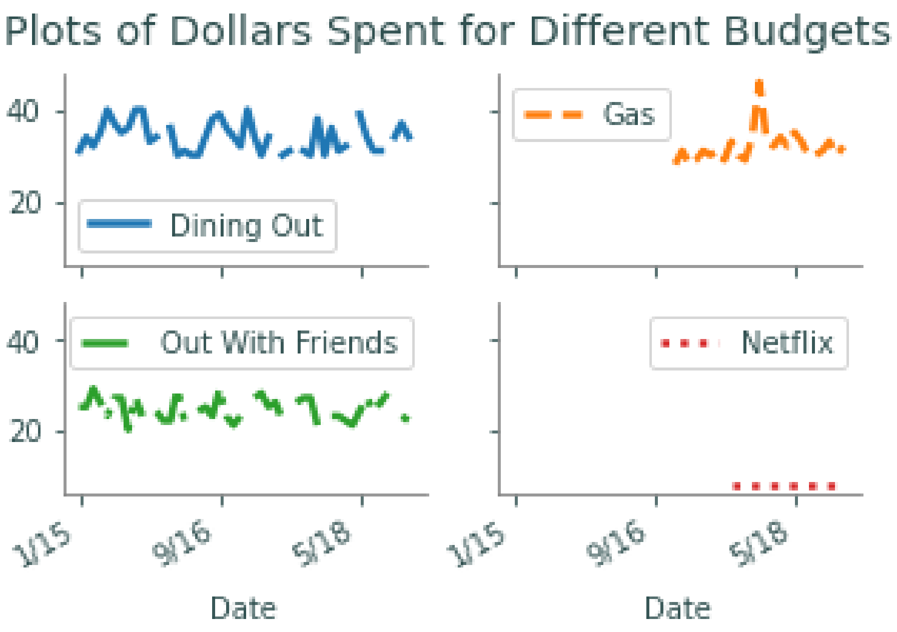
One way to declutter a visualization is to examine less data. Notice that `'Living Expenses'` has values much bigger than the other columns. Dropping this column, as well as `'Rent'`, gives a better overview of the data, shown in Figure 4.1b.

ACHTUNG!

Often plotting all data at once is unwise because columns have **different units of measure**. Be careful not to plot parts of a data set together if those parts do not have the same units or are otherwise incomparable.

Another way to declutter a plot is to use subplots. To quickly plot several columns in separate subplots, use `subplots=True` and specify a shape tuple as the `layout` for the plots. Subplots automatically share the same x-axis. Set `sharey=True` to force them to share the same y-axis as well.

```
>>> budget.plot(y=['Dining Out','Gas','Out With Friends', 'Netflix'],
...     subplots=True, layout=(2,2), sharey=True,
...     style=['-','--','-.',':'],title="Plots of Dollars Spent for Different ↩
   Budgets")
```



As mentioned previously, the `plot()` method can be used to plot different kinds of plots. One possible kind of plot is a histogram. Since plots made by the `plot()` method share an x-axis by default, histograms turn out poorly whenever there are columns with very different data ranges or when more than one column is plotted at once.

```
# Plot three histograms together.
>>> budget.plot(kind='hist',y=['Gas','Dining Out','Out With Friends'],
...     alpha=.7,bins=10,title="Frequency of Amount (in dollars) Spent")

# Plot three histograms, stacking one on top of the other.
>>> budget.plot(kind='hist',y=['Gas','Dining Out','Out With Friends'],
...     bins=10,stacked=True,title="Frequency of Amount (in dollars) Spent")
```

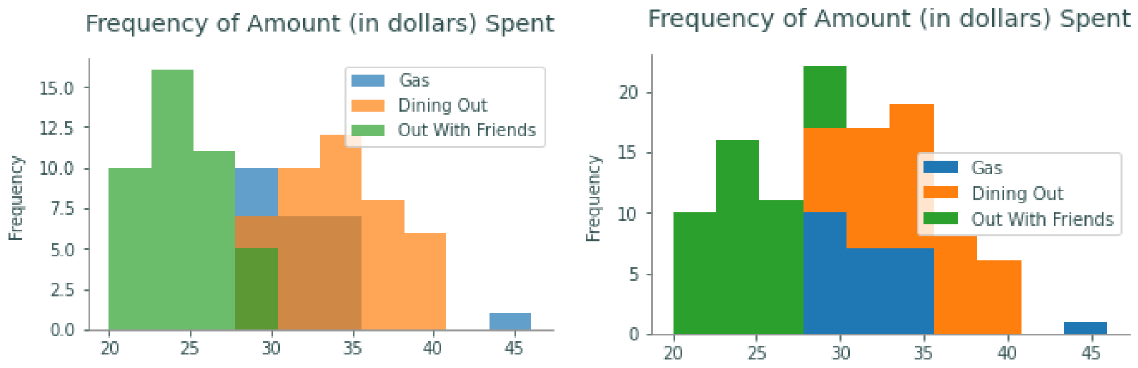


Figure 4.2: Two examples of histograms that are difficult to understand because multiple columns are plotted.

Thus, histograms are good for examining the distribution of a **single** column in a data set. For histograms, use the `hist()` method of the `DataFrame` instead of the `plot()` method. Specify the number of bins with the `bins` parameter. Choose a number of bins that accurately represents the data; the wrong number of bins can create a misleading or uninformative visualization.

```
>>> budget[["Dining Out","Gas"]].hist(grid=False,bins=10)
```

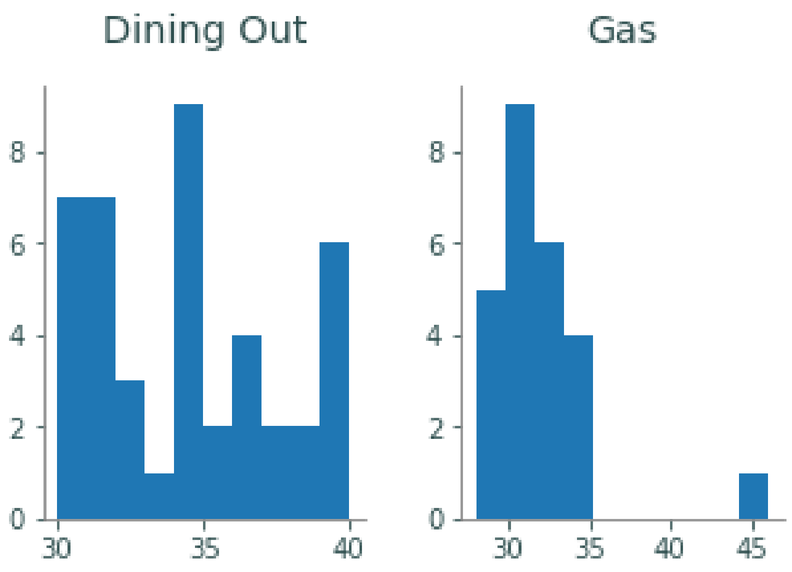


Figure 4.3: Histograms of "Dining Out" and "Gas".

Problem 1. Create 3 visualizations for the data in `crime_data.csv`. Make one of the visualizations a histogram. The visualizations should be well labeled and easy to understand. Include a short description of your plots as a caption.

Patterns and Correlations

After visualizing the entire data set initially, a good next step is to closely compare related parts of the data. This can be done with different types of visualizations. For example, Figure 4.1b suggests that the "Dining Out" and "Out With Friends" columns are roughly on the same scale. Since this data is sequential (indexed by time), start by plotting these two columns against the index. Next, create a scatter plot of one of the columns versus the other to investigate correlations that are independent of the index. Unlike other types of plots, using `kind="scatter"` requires both `x` and `y` columns as arguments.

```
# Plot 'Dining Out' and 'Out With Friends' as lines against the index.
>>> budget.plot(y=["Dining Out", "Out With Friends"],title="Amount Spent on ↩
    Dining Out and Out with Friends per Day")
```

```
# Make a scatter plot of 'Dining Out' against 'Out With Friends'
>>> budget.plot(kind="scatter", x="Dining Out", y="Out With Friends",
...     alpha=.8,xlim=(0,max(budget['Dining Out'])+1),
...     ylim=(0,max(budget['Out With Friends'])+1))
```

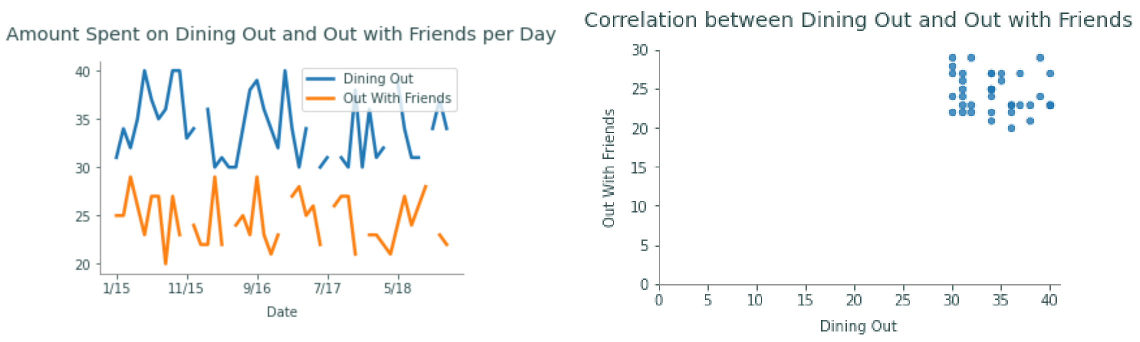


Figure 4.4: Correlations between "Dining Out" and "Out With Friends".

The first plot shows us that more money is spent on dining out than being out with friends overall. However, both categories stay in the same range for most of the data. This is confirmed in the scatter plot by the block in the upper right corner, indicating the common range spent on dining out and being out with friends.

Achtung!

When analyzing data, especially while searching for patterns and correlations, **always** ask yourself if the data makes sense and is trustworthy. What lurking variables could have influenced the data measurements as they were being gathered?

The crime data set from Problem 1 is somewhat suspect in this regard. The murder rate is likely accurate, since murder is conspicuous and highly reported, but what about the rape rate? Are the number of rapes increasing, or is the percentage of rapes being reported increasing? It's probably both! Be careful about drawing conclusions for sensitive or questionable data.

Another useful visualization used to understand correlations in a data set is a scatter matrix. The function pd.plotting.scatter_matrix() produces a table of plots where each column is plotted against each other column in separate scatter plots. The plots on the diagonal, instead of plotting a column against itself, displays a histogram of that column. This provides a very quick method for an initial analysis of the correlation between different columns.

```
>>> pd.plotting.scatter_matrix(budget[['Living Expenses','Other']])
```

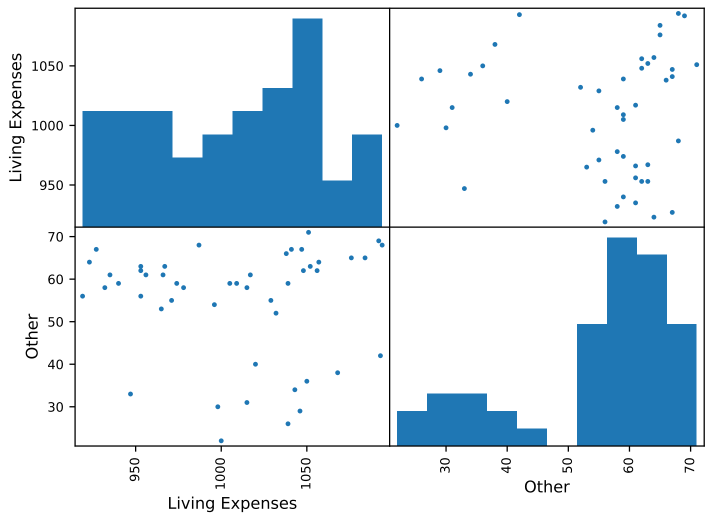


Figure 4.5: Scatter matrix comparing "Living Expenses" and "Other".

Bar Graphs

Different types of graphs help to identify different patterns. Note that the data set budget gives monthly expenses. It may be beneficial to look at one specific month. Bar graphs are a good way to compare small portions of the data set.

As a general rule, horizontal bar charts (kind="barh") are better than the default vertical bar charts (kind="bar") because most humans can detect horizontal differences more easily than vertical differences. If the labels are too long to fit on a normal figure, use plt.tight_layout() to adjust the plot boundaries to fit the labels in.

```
# Plot all data for the last month in the budget
>>> budget.iloc[-1,:].plot(kind='barh')
>>> plt.tight_layout()

# Plot all data for the last month without 'Rent' and 'Living Expenses'
>>> budget.drop(['Rent','Living Expenses'],axis=1).iloc[-1,:].plot(kind='barh')
>>> plt.tight_layout()
```

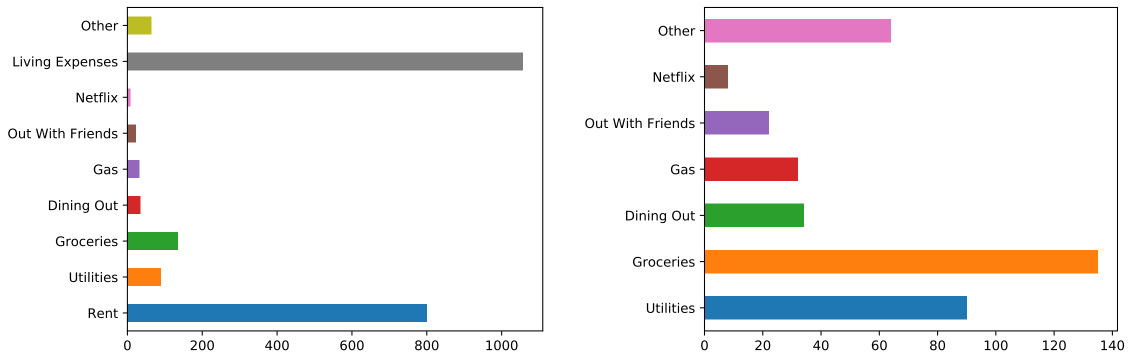


Figure 4.6: Bar graphs showing expenses paid in the last month of `budget`.

Problem 2. Using the crime data from the previous problem, identify if a trend exists between `Forcible Rape` and the following variables:

1. `Violent`
2. `Burglary`
3. `Aggravated Assault`

Make sure each graph is clearly labelled and readable. Include a caption explaining whether there is a visual trend between the variables.

Distributional Visualizations

While histograms are good at displaying the distributions for one column, a different visualization is needed to show the distribution of an entire set. A *box plot*, sometimes called a "cat-and-whisker" plot, shows the five number summary: the minimum, first quartile, median, third quartile, and maximum of the data. Box plots are useful for comparing the distributions of relatable data. However, box plots are a basic summary, meaning that they are susceptible to miss important information such as how many points were in each distribution.

```
# Compare the distributions of four columns.
>>> budget.plot(kind="box", y=["Gas","Dining Out","Out With Friends","Other"])

# Compare the distributions of all columns but 'Rent' and 'Living Expenses'.
>>> budget.drop(["Rent", "Living Expenses"], axis=1).plot(kind="box",
...     vert=False)
```

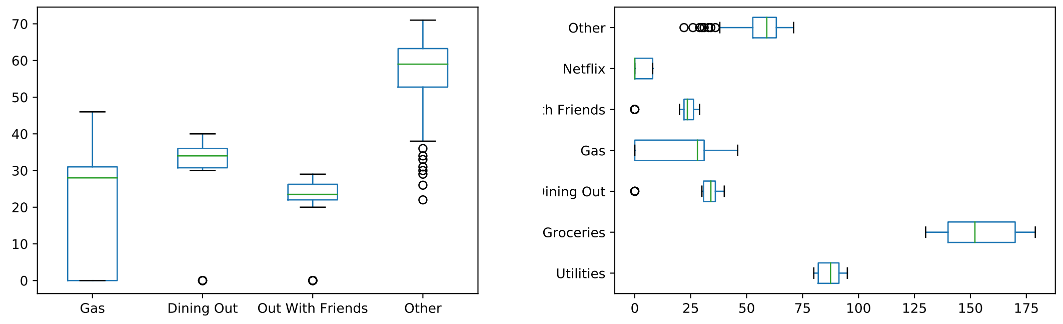


Figure 4.7: Vertical and horizontal box plots of `budget` dataset.

Hexbin Plots

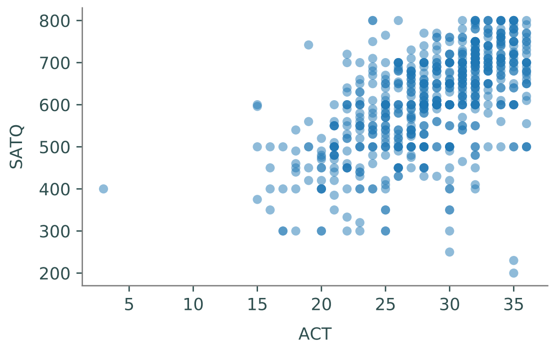
A scatter plot is essentially a plot of samples from the joint distribution of two columns. However, scatter plots can be uninformative for large data sets when the points in a scatter plot are closely clustered. *Hexbin plots* solve this problem by plotting point density in hexagonal bins—essentially creating a 2-dimensional histogram.

The file `sat_act.csv` contains 700 self reported scores on the SAT Verbal, SAT Quantitative and ACT, collected as part of the Synthetic Aperture Personality Assessment (SAPA) web based personality assessment project. The obvious question with this data set is "how correlated are ACT and SAT scores?" The scatter plot of ACT scores versus SAT Quantitative scores, Figure 4.8a, is highly cluttered, even though the points have some transparency. A hexbin plot of the same data, Figure 4.8b, reveals the **frequency** of points in binned regions.

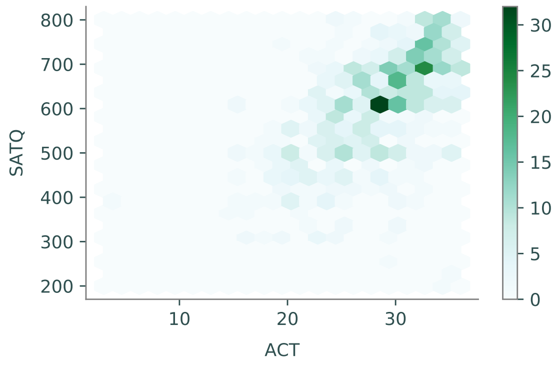
```
>>> satact = pd.read_csv("sat_act.csv", index_col="ID")
>>> list(satact.columns)
['gender', 'education', 'age', 'ACT', 'SATV', 'SATQ']

# Plot the ACT scores against the SAT Quant scores in a regular scatter plot.
>>> satact.plot(kind="scatter", x="ACT", y="SATQ", alpha=.8)

# Plot the densities of the ACT vs. SATQ scores with a hexbin plot.
>>> satact.plot(kind="hexbin", x="ACT", y="SATQ", gridsize=20)
```



(a) ACT vs. SAT Quant scores.



(b) Frequency of ACT vs. SAT Quant scores.

Figure 4.8: Scatter plots and hexbin plot of SAT and ACT scores.

Just as choosing a good number of `bins` is important for a good histogram, choosing a good `gridsize` is crucial for an informative hexbin plot. A large `gridsize` creates many small bins and a small `gridsize` creates fewer, larger bins.

Note

Since hexbins are based on frequencies, they are prone to being misleading if the dataset is not understood well. For example, when plotting information that deals with geographic position, increases in frequency may be results in higher populations rather than the actual information being plotted.

See `http://pandas.pydata.org/pandas-docs/stable/visualization.html` for more types of plots available in Pandas and further examples.

Problem 3. Use `crime_data.csv` to display the following distributions.

1. The distributions of `Burglary`, `Violent`, and `Vehicle Theft`,
2. The distributions of `Vehicle Thefts` against the values of `Robbery`.

As usual, all plots should be labeled and easy to read.

Hint: To get the x-axis label to display, you might need to set the sharex parameter of plot() to False.

Principles of Good Data Visualization

Data visualization is a powerful tool for analysis and communication. When writing a paper or report, the author must make many decisions about how to use graphics effectively to convey useful information to the reader. Here we will go over a simple process for making deliberate, effective, and efficient design decisions.

Attention to Detail

Consider the plot in Figure 4.9. It is a scatter plot of positively correlated data of some kind, with `temp`–likely temperature–on the x axis and `cons` on the y axis. However, the picture is not really communicating anything about the dataset. It has not specified the units for the x or the y axis, nor does it tell what `cons` is. There is no title, and the source of the data is unknown.

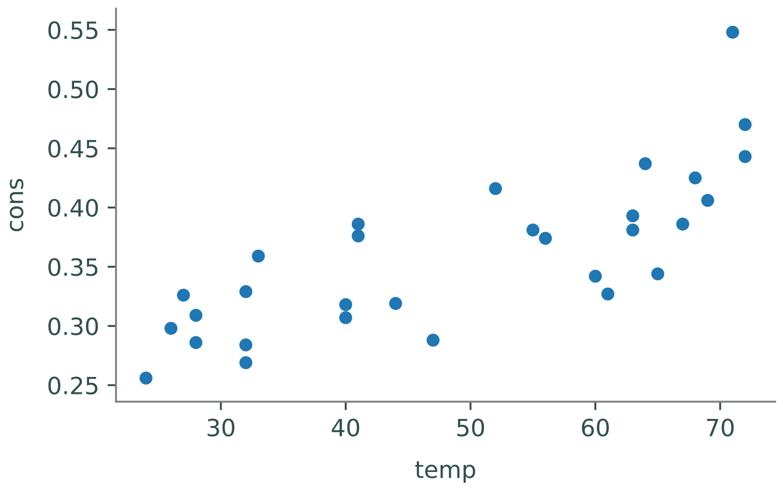


Figure 4.9: Non-specific data.

Labels and Citations

In a homework or lab setting, we sometimes (mistakenly) think that it is acceptable to leave off appropriate labels, legends, titles, and sourcing. In a published report or presentation, this kind of carelessness is confusing at best and, when the source is not included, even plagiaristic. Data needs to be explained in a useful manner that includes all of the vital information.

Consider again Figure 4.9. This figure comes from the `Icecream` dataset within the `pydataset` package, which we store here in a dataframe and then plot:

```
>>> from pydataset import data
>>> icecream = data("Icecream")
>>> icecream.plot(kind="scatter", x="temp", y="cons")
```

This code produces the rather substandard plot in Figure 4.9. Examining the source of the dataset can give important details to create better plots. When plotting data, make sure to understand what the variable names represent and where the data was taken from. Use this information to create a more effective plot.

The ice cream data used in Figure 4.9 is better understood with the following information:

1. The dataset details ice cream consumption via 30 four-week periods from March 1951 to July 1953 in the United States.
2. `cons` corresponds to "consumption of ice cream per capita" and is measured in pints.
3. `income` is the family's weekly income in dollars.
4. `price` is the price of a pint of ice cream.
5. `temp` corresponds to temperature, degrees Fahrenheit.
6. The listed source is: "Hildreth, C. and J. Lu (1960) *Demand relations with autocorrelated disturbances*, Technical Bulletin No 2765, Michigan State University."

This information gives important details that can be used in the following code. As seen in previous examples, pandas automatically generates legends when appropriate. Pandas also automatically labels the x and y axes, however our data frame column titles may be insufficient. Appropriate titles for the x and y axes must also list appropriate units. For example, the y axis should specify that the consumption is in units of *pints per head*, in place of the ambiguous label `cons`.

```
>>> icecream = data("Icecream")
# Set title via the title keyword argument
>>> icecream.plot(kind="scatter", x="temp", y="cons",
...     title="Ice Cream Consumption in the U.S., 1951-1953")
# Override pandas automatic labelling using xlabel and ylabel
>>> plt.xlabel("Temp (Fahrenheit)")
>>> plt.ylabel("Consumption per head (pints)")
```

To add the necessary text to the figure, use either `plt.annotate()` or `plt.text()`. Alternatively, add text immediately below wherever the figure is displayed. The first two parameters of `plt.text` are the x and y coordinates to place the text. The third parameter is the text to write. For instance, using `plt.text(0.5, 0.5, "Hello World")` will center the `Hello World` string in the axes.

```
>>> plt.text(20, .1, r"Source: Hildreth, C. and J. Lu (1960) \emph{Demand"
...     "relations with autocorrelated disturbances}\nTechnical Bulletin No"
...     "2765, Michigan State University.", fontsize=7)
```

Both of these methods are imperfect but can normally be easily replaced by a caption attached to the figure. Again, we reiterate how important it is that you source any data you use; failing to do so is plagiarism.

Finally, we have a clear and demonstrative graphic in Figure 4.10.

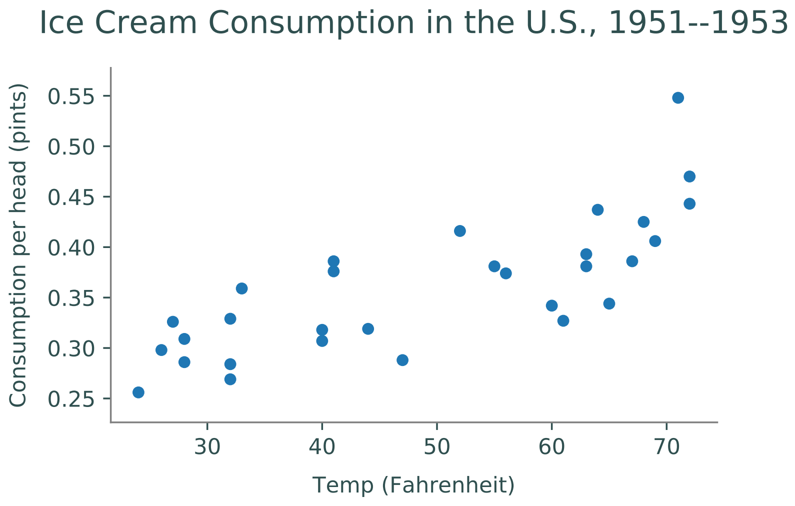


Figure 4.10: Source: Hildreth, C. and J. Lu (1960) *Demand relations with autocorrelated disturbances*, Technical Bulletin No 2765, Michigan State University.

ACHTUNG!

Visualizing data can inherit many biases of the visualizer and as a result can be intentionally misleading. Examples of this include, but are not limited to, visualizing subsets of data that do not represent the whole of the data and having purposely misconstrued axes. Every data visualizer has the responsibility to avoid including biases in their visualizations to ensure data is being represented informatively and accurately.

Problem 4. The dataset `college.csv` contains information from 1995 on universities in the United States. To access information on variable names, go to `https://cran.r-project.org/web/packages/ISLR/ISLR.pdf`. Create 3 plots that compare variables or universities. These plots should answer questions about the data, e.g. what is the distribution of graduation rates or do schools with lower student to faculty ratios have higher tuition costs. These three plots should be easy to understand and have clear variable names and citations.

5 Numerical Methods for Initial Value Problems; Harmonic Oscillators

Lab Objective: *Implement several basic numerical methods for initial value problems (IVPs) and use them to study harmonic oscillators.*

Methods for Initial Value Problems

Consider the *initial value problem* (IVP)

$$\begin{aligned} \mathbf{x}'(t) &= f(\mathbf{x}(t), t), \quad t_0 \leq t \leq t_f \\ \mathbf{x}(t_0) &= \mathbf{x}_0, \end{aligned} \tag{5.1}$$

where f is a suitably continuous function. A solution of (5.1) is a continuously differentiable, and possibly vector-valued, function $\mathbf{x}(t) = [x_1(t), \ldots, x_m(t)]^\mathsf{T}$, whose derivative $\mathbf{x}'(t)$ equals $f(\mathbf{x}(t), t)$ for all $t \in [t_0, t_f]$, and for which the *initial value* $\mathbf{x}(t_0)$ equals $\mathbf{x}_0$.

Under the right conditions, namely that f is uniformly Lipschitz continuous in $\mathbf{x}(t)$ near $\mathbf{x}_0$ and continuous in t near t_0, (5.1) is well-known to have a unique solution. However, for many IVPs, it is difficult, if not impossible, to find a closed-form, analytic expression for $\mathbf{x}(t)$. In these cases, numerical methods can be used to instead *approximate* $\mathbf{x}(t)$.

As an example, consider the initial value problem

$$\begin{aligned} x'(t) &= \sin(x(t)), \\ x(0) &= x_0. \end{aligned} \tag{5.2}$$

The solution $x(t)$ is defined implicitly by

$$t = \ln \left| \frac{\cos(x_0) + \cot(x_0)}{\csc(x(t)) + \cot(x(t))} \right|.$$

This equation cannot be solved for $x(t)$, so it is difficult to understand what solutions to (5.2) look like. Since $sin(n\pi) = 0$, there are constant solutions $x_n(t) = n\pi$, $n \in \mathbb{Z}$. Using a numerical IVP solver, solutions for different values of x_0 can be approximated. Figure 5.1 shows several of these approximate solutions, along with some of the constant, or *equilibrium*, solutions.

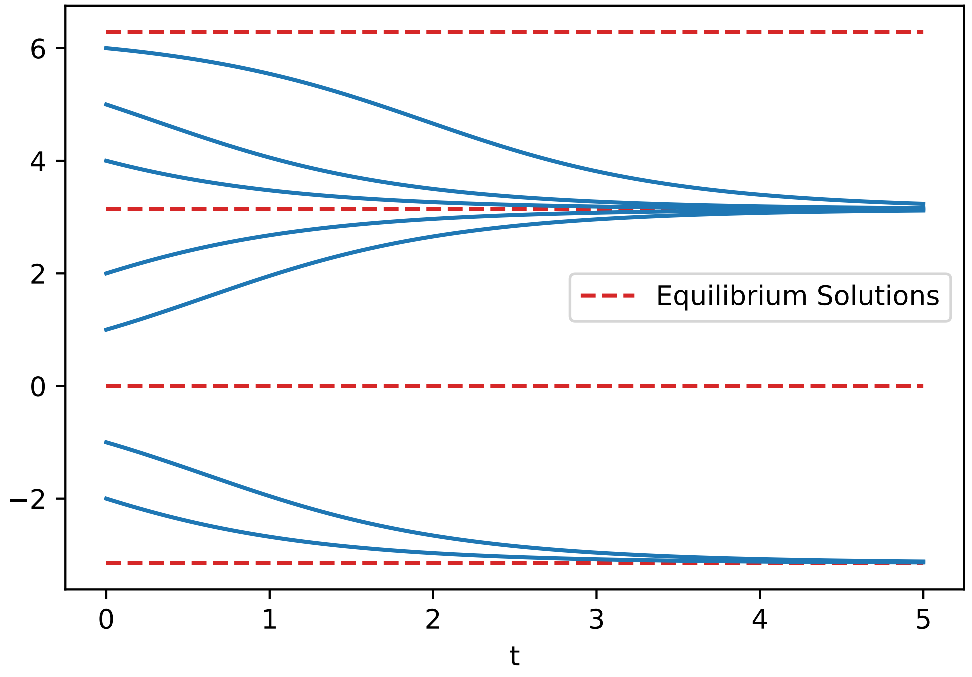


Figure 5.1: Several solutions of (5.2), using `scipy.integrate.odeint`.

Numerical Methods

For the numerical methods that follow, the key idea is to seek an approximation for the values of $\mathbf{x}(t)$ only on a finite set of values $t_0 < t_1 < \ldots < t_{n-1} < t_n \ (= t_f)$. In other words, these methods try to solve for $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ such that $\mathbf{x}_i \approx \mathbf{x}(t_i)$.

Euler's Method

For simplicity, assume that each of the n subintervals $[t_{i-1}, t_i]$ has equal length $h = (t_f - t_0)/n$. h is called the *step size*. Assuming $\mathbf{x}(t)$ is twice-differentiable, for each component function $x_j(t)$ of $\mathbf{x}(t)$ and for each i, Taylor's Theorem says that

$$x_j(t_{i+1}) = x_j(t_i) + hx_j'(t_i) + \frac{h^2}{2}x_j''(c) \text{ for some } c \in [t_i, t_{i+1}].$$

The quantity $\frac{h^2}{2}x_j''(c)$ is negligible when h is sufficiently small, and thus $x_j(t_{i+1}) \approx x_j(t_i) + hx_j'(t_i)$. Therefore, bringing the component functions of $\mathbf{x}(t)$ back together gives

$$\begin{aligned}\mathbf{x}(t_{i+1}) &\approx \mathbf{x}(t_i) + h\mathbf{x}'(t_i),\\ &\approx \mathbf{x}(t_i) + hf(\mathbf{x}(t_i), t_i).\end{aligned}$$

This approximation leads to the *Euler method*: Starting with $\mathbf{x}_0 = \mathbf{x}(t_0)$, $\mathbf{x}_{i+1} = \mathbf{x}_i + hf(\mathbf{x}_i, t_i)$ for $i = 0, 1, \ldots, n-1$. Euler's method can be understood as starting with the point at $\mathbf{x}_0$, then calculating the derivative of $\mathbf{x}(t)$ at t_0 using $f(\mathbf{x}_0, t_0)$, followed by taking a step in the direction of the derivative scaled by h. Set that new point as $\mathbf{x}_1$ and continue.

It is important to consider how the choice of step size h affects the accuracy of the approximation. Note that at each step of the algorithm, the *local truncation error*, which comes from neglecting the $x_j''(c)$ term in the Taylor expansion, is proportional to h^2. The error $||\mathbf{x}(t_i) - \mathbf{x}_i||$ at the *ith* step comes from $i = \frac{t_i - t_0}{h}$ steps, which is proportional to h^{-1}, each contributing h^2 error. Thus the *global truncation error* is proportional to h. Therefore, the Euler method is called a *first-order method*, or a $\mathcal{O}(h)$ method. This means that as h gets small, the approximation of $\mathbf{x}(t)$ improves in two ways. First, $\mathbf{x}(t)$ is approximated at more values of t (more information about the solution), and second, the accuracy of the approximation at any t_i is improved proportional to h (better information about the solution).

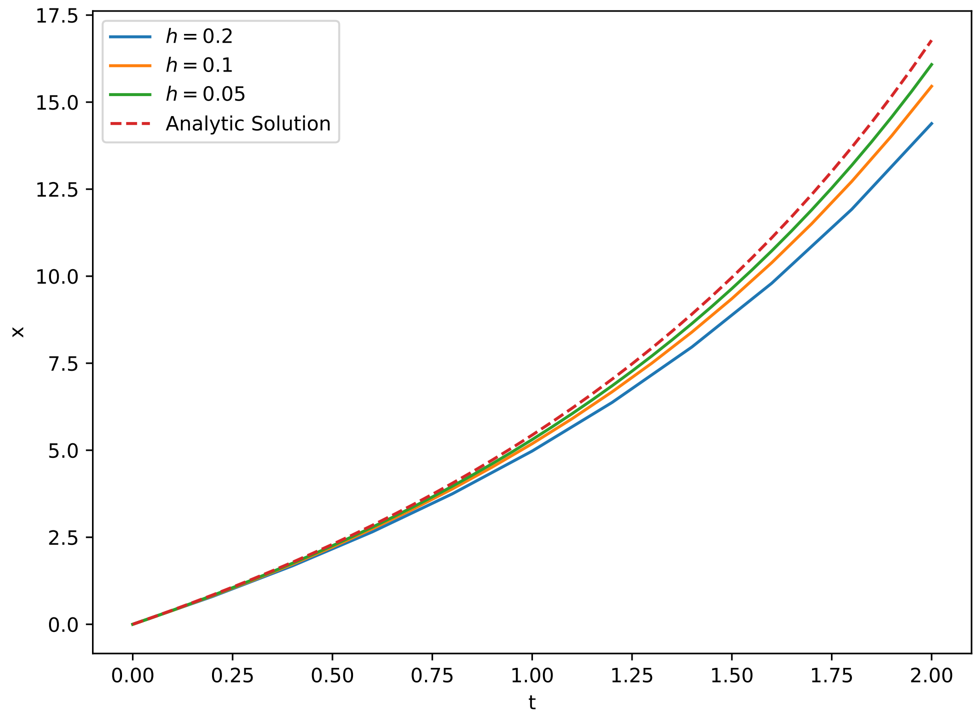


Figure 5.2: The solution of (5.3), alongside several approximations using Euler's method.

Problem 1. Write a function which implements Euler's method for an IVP of the form (5.1). Test your function on the IVP:

$$\begin{aligned} x'(t) &= x(t) - 2t + 4, \quad 0 \le t \le 2, \\ x(0) &= 0, \end{aligned} \tag{5.3}$$

where the analytic solution is $x(t) = -2 + 2t + 2e^t$. Use the Euler method to numerically approximate the solution with step sizes $h = 0.2, 0.1$, and 0.05. Plot the true solution alongside the three approximations, and compare your results with Figure 5.2.

Midpoint Method

The midpoint method is very similar to Euler's method. For small h, use the approximation

$$\mathbf{x}(t_{i+1}) \approx \mathbf{x}(t_i) + hf(\mathbf{x}(t_i) + \frac{h}{2}f(\mathbf{x}(t_i), t_i), t_i + \frac{h}{2},).$$

In this approximation, first set $\hat{\mathbf{x}}_i = \mathbf{x}_i + \frac{h}{2}f(\mathbf{x}_i, t_i)$, which is an Euler method step of size $h/2$. Then evaluate $f(\hat{\mathbf{x}}_i, t_i + \frac{h}{2})$, which is a more accurate approximation to the derivative $\mathbf{x}'(t)$ in the interval $[t_i, t_{i+1}]$. Finally, a step is taken in that direction, scaled by h. It can be shown that the local truncation error for the midpoint method is $\mathcal{O}(h^3)$, giving global truncation error of $\mathcal{O}(h^2)$. This is a significant improvement over the Euler method. However, it comes at the cost of additional evaluations of f and a handful of extra floating point operations on the side. This tradeoff will be considered later in the lab.

Runge-Kutta Methods

The Euler method and the midpoint method belong to a family called *Runge-Kutta methods*. There are many Runge-Kutta methods with varying orders of accuracy. Methods of order four or higher are most commonly used. A fourth-order Runge-Kutta method (RK4) iterates as follows:

$$\begin{aligned}
K_1 &= f(\mathbf{x}_i, t_i),\\
K_2 &= f(\mathbf{x}_i + \frac{h}{2}K_1, t_i + \frac{h}{2}),\\
K_3 &= f(\mathbf{x}_i + \frac{h}{2}K_2, t_i + \frac{h}{2}),\\
K_4 &= f(\mathbf{x}_i + hK_3, t_{i+1}),\\
\mathbf{x}_{i+1} &= \mathbf{x}_i + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4).
\end{aligned}$$

Runge-Kutta methods can be understood as a generalization of quadrature methods for approximating integrals, where the integrand is evaluated at specific points, and then the resulting values are combined in a weighted sum. For example, consider a differential equation

$$x'(t) = f(t)$$

Since the function f has no x dependence, this is a simple integration problem. In this case, Euler's method corresponds to the left-hand rule, the midpoint method becomes the midpoint rule, and RK4 reduces to Simpson's rule.

Advantages of Higher-Order Methods

It can be useful to visualize the order of accuracy of a numerical method. A method of order p has relative error of the form

$$E(h) = Ch^p$$

taking the logarithm of both sides yields

$$log(E(h)) = p \cdot log(h) + log(C)$$

Therefore, on a log-log plot against h, $E(h)$ is a line with slope p and intercept $log(C)$.

Problem 2. Write functions that implement the midpoint and fourth-order Runge-Kutta methods. Use the Euler, Midpoint, and RK4 methods to approximate the value of the solution for the IVP (5.3) from Problem 1 for step sizes of $h = 0.2$, 0.1, 0.05, 0.025, and 0.0125.

Plot the following graphs

- The true solution alongside the approximation obtained from each method when $h = 0.2$.
- A log-log plot (use `plt.loglog`) of the relative error $|x(2) - x_n|/|x(2)|$ as a function of h for each approximation.

Compare your second plot with Figure 5.3.

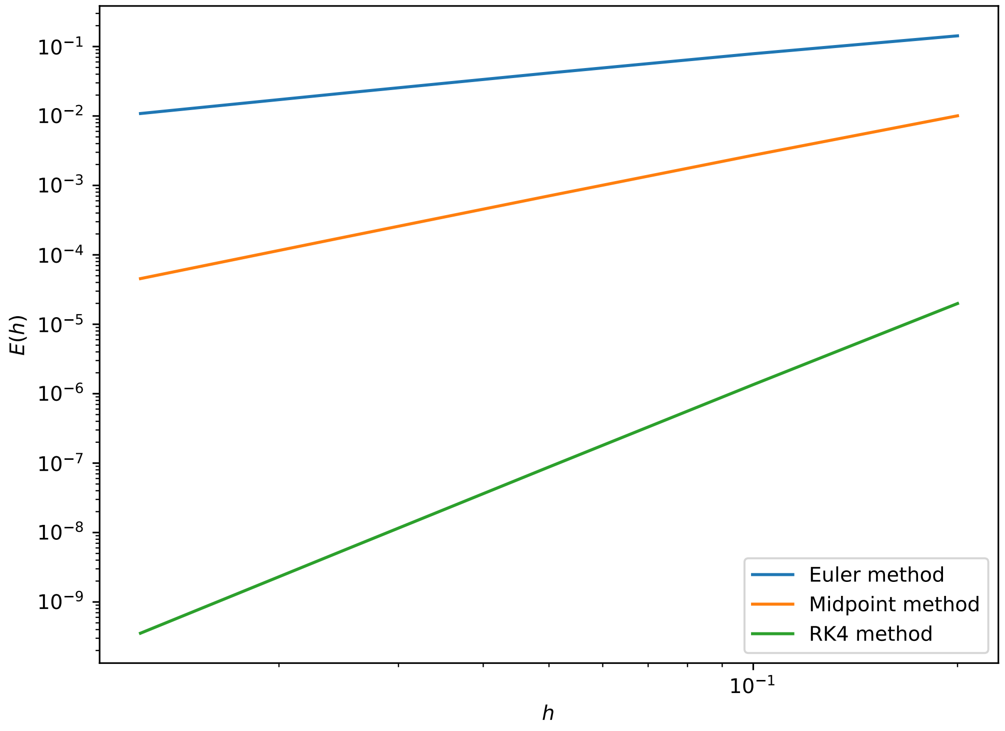


Figure 5.3: Loglog plot of the relative error in approximating $x(2)$, using step sizes $h = 0.2$, 0.1, 0.05, 0.025, and 0.0125. The slope of each line demonstrates the first, second, and fourth order convergence of the Euler, Midpoint, and RK4 methods, respectively.

The Euler, midpoint, and RK4 methods help illustrate the potential trade-off between order of accuracy and computational expense. To increase the order of accuracy, more evaluations of f must be performed at each step. It is possible that this trade-off could make higher-order methods undesirable, as (in theory) one could use a lower-order method with a smaller step size h. However, this is not generally the case. Assuming efficiency is measured in terms of the number of f-evaluations required to reach a certain threshold of accuracy, higher-order methods turn out to be much more efficient. For example, consider the IVP

$$\begin{aligned} x'(t) &= x(t)\cos(t), \quad t \in [0, 8], \\ x(0) &= 1. \end{aligned} \tag{5.4}$$

Figure 5.4 illustrates the comparative efficiency of the Euler, Midpoint, and RK4 methods applied to (5.4). The higher-order RK4 method requires fewer f-evaluations to reach the same level of relative error as the lower-order methods. As h becomes small, which corresponds to increasing functional evaluations, each method reaches a point where the relative error $|x(8) - x_n|/|x(8)|$ stops improving. This occurs when h is so small that floating point round-off error overwhelms local truncation error. Notice that the higher-order methods are able to reach a better level of relative error before this phenomena occurs.

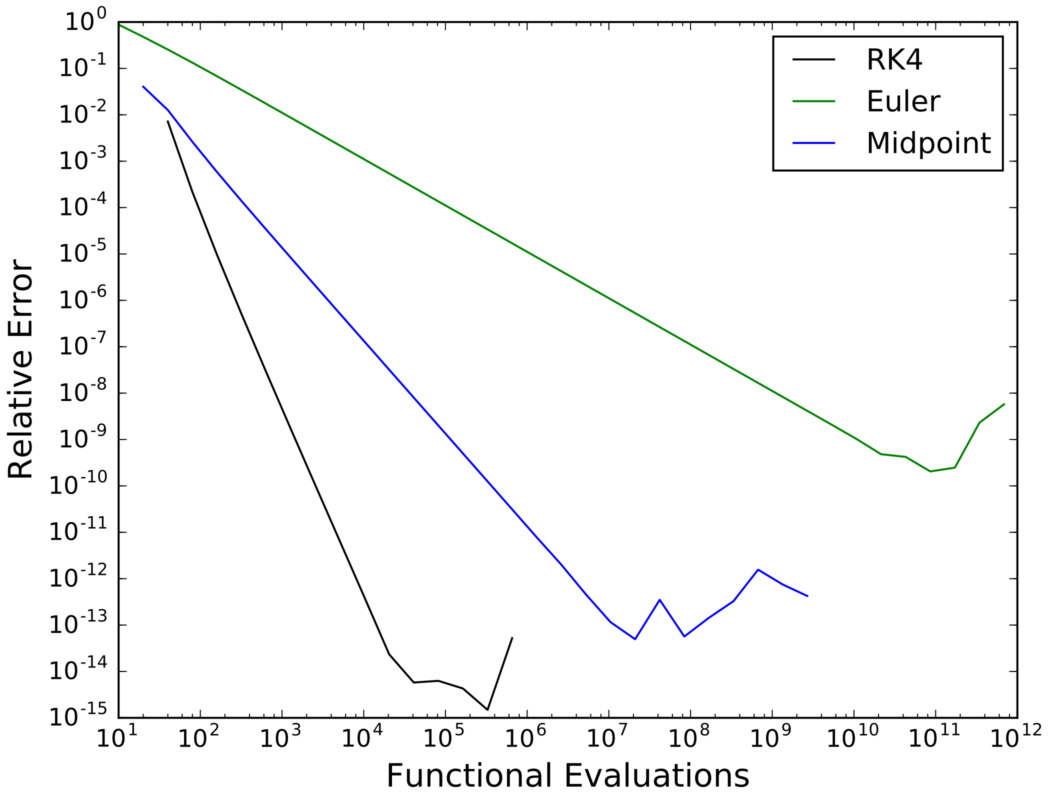


Figure 5.4: The relative error in computing the solution of (5.4) at $x = 8$ versus the number of times the right-hand side of (5.4) must be evaluated.

Harmonic Oscillators and Resonance

Harmonic oscillators are common in classical mechanics. A few examples include the pendulum (with small displacement), spring-mass systems, and the flow of electric current through various types of circuits. A harmonic oscillator $y(t)$[1] is a solution to an initial value problem of the form

$$\begin{aligned} &my'' + \gamma y' + ky = f(t), \\ &y(0) = y_0, \quad y'(0) = y_0'. \end{aligned}$$

Here, m represents the mass on the end of a spring, γ represents the effect of damping on the motion, k is the spring constant, and $f(t)$ is the external force applied.

Simple harmonic oscillators

A simple harmonic oscillator is a harmonic oscillator that is not damped, $\gamma = 0$, and is free, $f = 0$, rather than forced, $f \neq 0$. A simple harmonic oscillator can described by the IVP

$$\begin{aligned} &my'' + ky = 0, \\ &y(0) = y_0, \quad y'(0) = y_0'. \end{aligned}$$

The solution of this IVP is $y = c_1 \cos(\omega_0 t) + c_2 \sin(\omega_0 t)$, where $\omega_0 = \sqrt{k/m}$ is the natural frequency of the oscillator and c_1 and c_2 are determined by applying the initial conditions.

To solve this IVP using a Runge-Kutta method, it must be written in the form

$$\mathbf{x}'(t) = f(\mathbf{x}(t), t)$$

This can be done by setting $x_1 = y$ and $x_2 = y'$. Then we have

$$\mathbf{x}' = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}' = \begin{bmatrix} x_2 \\ \frac{-k}{m} x_1 \end{bmatrix}$$

Therefore

$$f(\mathbf{x}(t), t) = \begin{bmatrix} x_2 \\ \frac{-k}{m} x_1 \end{bmatrix}$$

Problem 3. Use the RK4 method to solve the simple harmonic oscillator satisfying

$$\begin{aligned} &my'' + ky = 0, \quad 0 \leq t \leq 20, \\ &y(0) = 2, \quad y'(0) = -1, \end{aligned} \tag{5.5}$$

for $m = 1$ and $k = 1$.

Plot your numerical approximation of $y(t)$. Compare this with the numerical approximation when $m = 3$ and $k = 1$. Consider: Why does the difference in solutions make sense physically?

[1] It is customary to write y instead of $y(t)$ when it is unambiguous that y denotes the dependent variable.

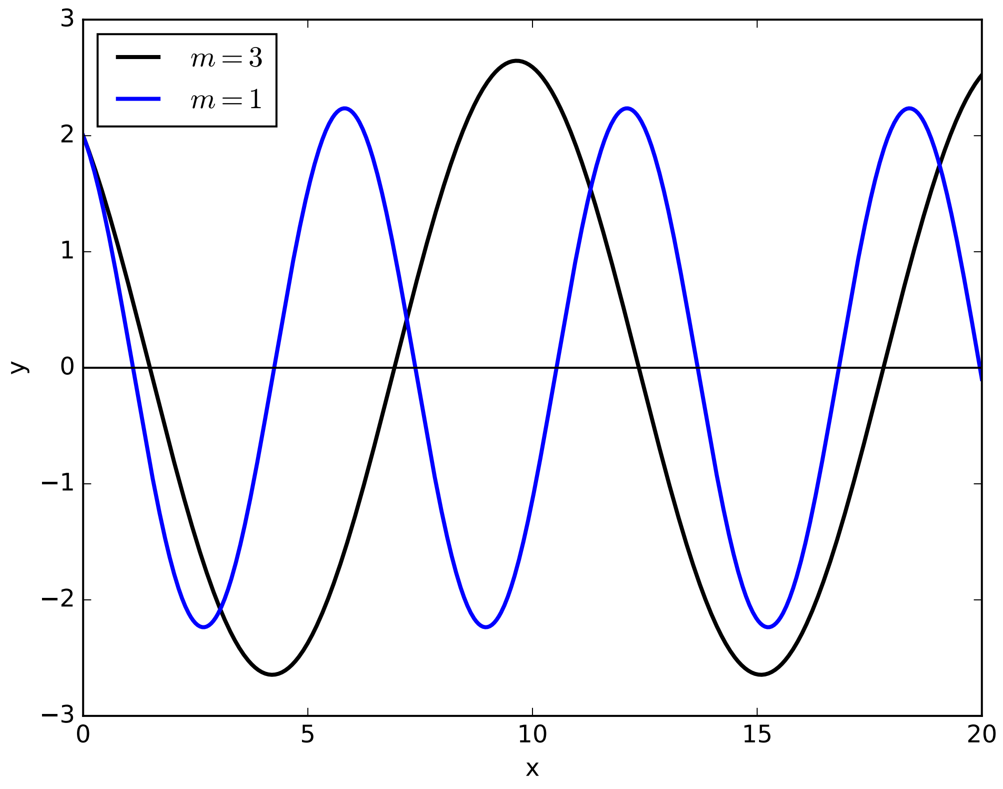


Figure 5.5: Solutions of (5.5) for several values of m.

Damped free harmonic oscillators

A damped free harmonic oscillator $y(t)$ satisfies the IVP

$$\begin{aligned} &my'' + \gamma y' + ky = 0, \\ &y(0) = y_0, \quad y'(0) = y'_0. \end{aligned}$$

The roots of the characteristic equation are

$$r_1, r_2 = \frac{-\gamma \pm \sqrt{\gamma^2 - 4km}}{2m}.$$

Note that the real parts of r_1 and r_2 are always negative, and so any solution $y(t)$ will decay over time due to a dissipation of the system energy. There are several cases to consider for the general solution of this equation:

1. If $\gamma^2 > 4km$, then the general solution is $y(t) = c_1e^{r_1t} + c_2e^{r_2t}$. Here the system is said to be *overdamped*. Notice from the general solution that there is no oscillation in this case.

2. If $\gamma^2 = 4km$, then the general solution is $y(t) = c_1e^{\gamma t/2m} + c_2te^{\gamma t/2m}$. Here the system is said to be *critically damped*.

3. If $\gamma^2 < 4km$, then the general solution is

$$\begin{aligned} y(t) &= e^{-\gamma t/2m}\left[c_1\cos(\mu t) + c_2\sin(\mu t)\right], \\ &= Re^{-\gamma t/2m}\sin(\mu t + \delta), \end{aligned}$$

where R and δ are fixed, and $\mu = \sqrt{4km - \gamma^2}/2m$. This system does oscillate.

Problem 4. Use the RK4 method to solve for the damped free harmonic oscillator satisfying

$$\begin{aligned} &y'' + \gamma y' + y = 0, \quad 0 \le t \le 20, \\ &y(0) = 1, \quad y'(0) = -1. \end{aligned}$$

For $\gamma = 1/2$, and $\gamma = 1$, simultaneously plot your numerical approximations of y.

Forced harmonic oscillators without damping

Consider the systems described by the differential equation

$$my''(t) + ky(t) = F(t). \tag{5.6}$$

In many instances, the external force $F(t)$ is periodic, so assume that $F(t) = F_0\cos(\omega t)$. If $\omega_0 = \sqrt{k/m} \neq \omega$, then the general solution of 5.6 is given by

$$y(t) = c_1\cos(\omega_0 t) + c_2\sin(\omega_0 t) + \frac{F_0}{m(\omega_0^2 - \omega^2)}\cos(\omega t).$$

If $\omega_0 = \omega$, then the general solution is

$$y(t) = c_1\cos(\omega_0 t) + c_2\sin(\omega_0 t) + \frac{F_0}{2m\omega_0}t\sin(\omega_0 t).$$

When $\omega_0 = \omega$, the solution contains a term that grows arbitrarily large as $t \to \infty$. If we included damping, then the solution would be bounded but large for small γ and ω close to ω_0.

Consider a physical spring-mass system. Equation 5.6 holds only for small oscillations; this is where Hooke's law is applicable. However, the fact that the equation predicts large oscillations suggests the spring-mass system could fall apart as a result of the external force. This mechanical resonance has been known to cause failure of bridges, buildings, and airplanes.

Problem 5. Use the RK4 method to solve the damped and forced harmonic oscillator satisfying

$$\begin{aligned} &2y'' + \gamma y' + 2y = 2\cos(\omega t), \quad 0 \le t \le 40, \\ &y(0) = 2, \quad y'(0) = -1. \end{aligned} \tag{5.7}$$

For the following values of γ and ω, plot your numerical approximations of $y(t)$: $(\gamma, \omega) = (0.5, 1.5)$, $(0.1, 1.1)$, and $(0, 1)$. Compare your results with Figure5.7.

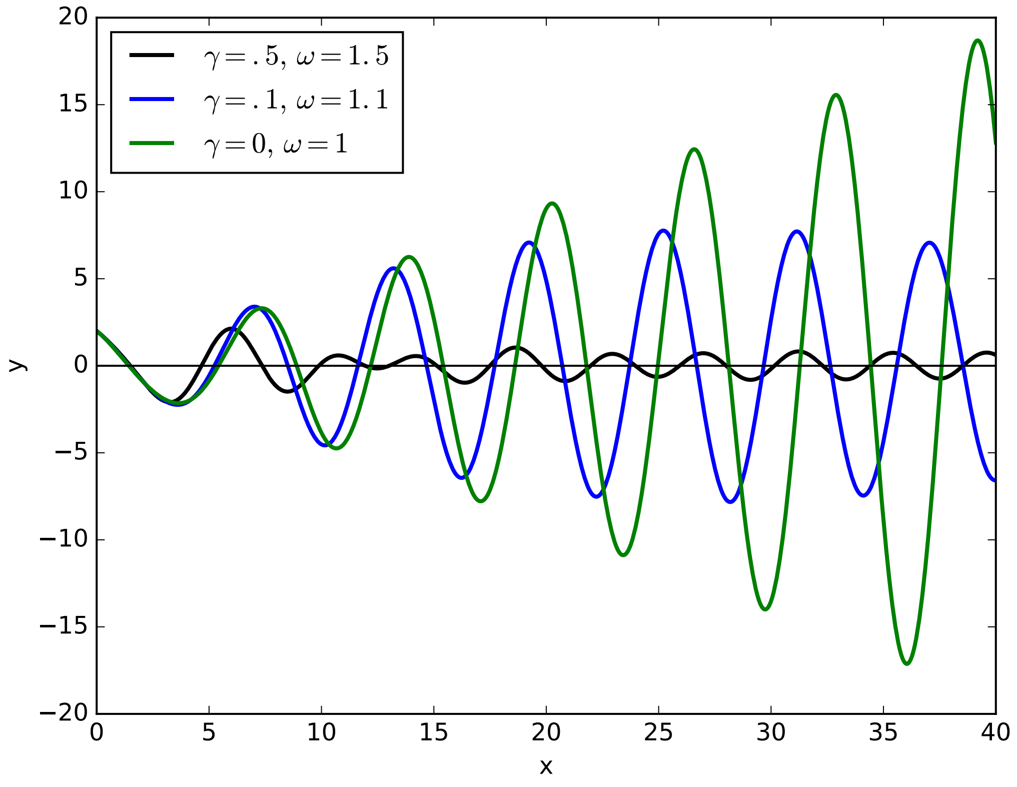


Figure 5.6: Solutions of (5.7) for several values of ω and γ.