

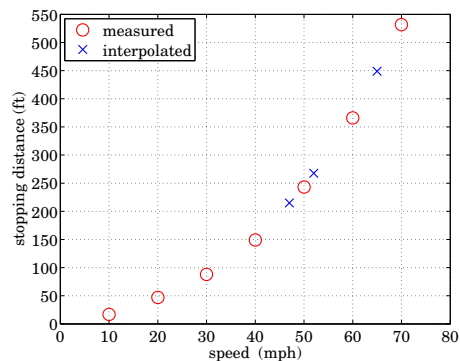
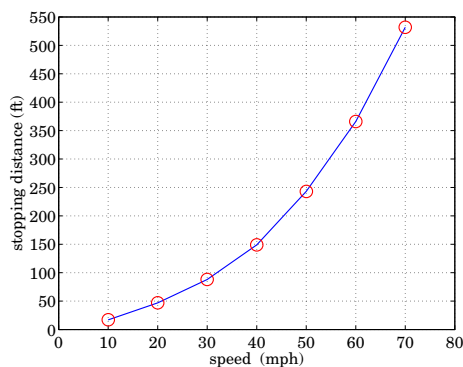
Homework Problems – Week 1
440:127 – Spring 2015 – S. J. Orfanidis

1. End-of-chapter Problem 2.14. Use 'format short e' to print your results.
2. Problem 2.15. Use 'format short e' to print your results.
3. Problem 2.18. Use 'format short e' to print your results. Revert to 'format' afterwards.
4. The NJ Driver's Manual contains information from which the following table can be obtained of stopping distances and stopping times of cars according to their traveling speeds:

speed (mph)	reaction distance (ft)	braking distance (ft)	total stopping distance (ft)	reaction time (sec)	deceleration (ft/sec ²)	total stopping time (sec)
10	11	6	17	0.75	17.89	1.57
20	22	25	47	0.75	17.25	2.45
30	33	55	88	0.75	17.60	3.25
40	44	105	149	0.75	16.39	4.33
50	55	188	243	0.75	14.29	5.88
60	66	300	366	0.75	12.90	7.57
70	77	455	532	0.75	11.59	9.61

- a. Suppose that the first column of speeds is given as well as the column of reaction times (5th column), and the column of total stopping times (7th column). From this information only, calculate the values shown in columns 2, 3, 4, and 6 of the above table for the reaction distances, braking distances, total stopping distances, and decelerations. Please use vectorized calculations only (no for-loops). Stopping means $v = 0$ and deceleration is negative acceleration.
- b. After reviewing the lecture notes (for week-2) and Section 7.2 of the text on the built-in function `fprintf`, please use `fprintf` to generate your output in a table that looks exactly as the one shown above. The numerical part of the table may be printed using `fprintf` in a loop, or by a single MATLAB command using the vectorized form of `fprintf` (but it's a good idea to figure out how to do both.)
- c. Make a plot of the total stopping distance versus the speed. Use appropriate MATLAB commands to annotate your graph as the one shown below.
- d. Based on the data in the above graph, use MATLAB to estimate the stopping distances for the following intermediate speed values $v_0 = [47, 52, 65]$ mph. On the graph, the data points are connected by straight lines.

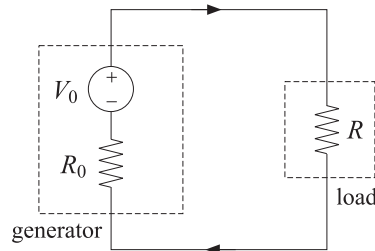
$$\left[\text{Hints: } 1 \text{ mile} = 5280 \text{ ft}, \quad v = v_0 + at, \quad s = v_0 t + \frac{1}{2} at^2 = \frac{(v_0 + at)^2 - v_0^2}{2a} = \frac{v^2 - v_0^2}{2a} \right]$$



5. Problem 2.11. Please use a vectorized calculation (no loops) to compute the amount of CO₂ (start with the vector of mpg's). Moreover, include appropriate MATLAB commands to print the results of your program in the format shown in the table below (you may use a for-loop here). [Hint: see the octaves example discussed in class, also note that the largest string in the second column has 21 characters, so use format %21s for that column.]

year	model	mpg	CO2
2008	smart for two	37 mpg	6291.89 lbs
2008	civic coupe	29 mpg	8027.59 lbs
2008	civic hybrid	43 mpg	5413.95 lbs
2008	chevrolet cobalt	30 mpg	7760.00 lbs
2008	toyota prius (hybrid)	46 mpg	5060.87 lbs
2008	toyota yaris	32 mpg	7275.00 lbs

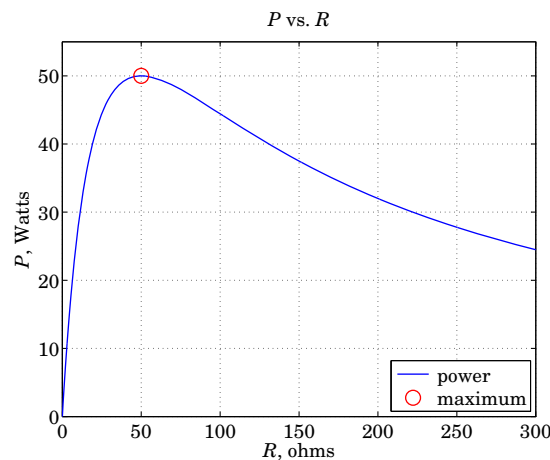
6. *Matched Loads*. An electric load (e.g., appliance, computer, etc.) connected to an electric power source (e.g., wall outlet, battery, etc.) appears to the power source as a resistance (impedance, more generally). Using Ohm's and Kirchoff's laws, it can be shown that the power delivered to and consumed by a load is given by:



$$P = \frac{V_0^2 R}{(R_0 + R)^2} \quad (1)$$

where R is the load resistance, and V_0, R_0 are the voltage of the generator and its internal resistance. The *maximum power transfer theorem* states that maximum power is delivered to the load when the load resistance matches that of the generator, that is, when $R = R_0$.

- For the following values $V_0 = 100$ volts, $R_0 = 50$ ohm, calculate the maximum power P that can be delivered to a matched load (the power will be in units of Watts if the voltage and resistances are in units of volts and ohms, respectively).
- Using the `linspace` function, generate a vector of 601 resistance values in the interval $0 \leq R \leq 300$ ohms, and using a *vectorized* version of Eq. (1), calculate the corresponding 601 values of P in units of Watts. Make a plot of P versus R , and indicate on the graph the point of maximum power transfer.
- Add appropriate MATLAB commands to reproduce all the graph annotations shown below, i.e., title, axis labels, axis limits, tickmarks, grid, and legends. [Hint: see the octaves example].



Note: The proof of the maximum power transfer theorem is straightforward. If you know calculus, you may compute the derivative dP/dR and set it equal to zero to obtain $R = R_0$. Alternatively, it's a matter of algebra to show the following identity in R :

$$P = \frac{V_0^2 R}{(R_0 + R)^2} = \frac{V_0^2}{4R_0} \cdot \left[1 - \frac{(R - R_0)^2}{(R + R_0)^2} \right]$$

which shows that for all values of R ,

$$P \leq \frac{V_0^2}{4R_0}$$

with equality reached at $R = R_0$.

Homework Problems – Week 2
440:127 – Spring 2015 – S. J. Orfanidis

1. Problem 3.3 of the text.
2. Problem 3.4 of the text.
3. Problem 3.5 of the text.
4. Problem 3.16 of the text. In addition, make a plot of the range versus the angle θ in degrees, and place on the graph your found maximum range and the angle at which it occurs. [*Hint*: use the function **max**]
5. The harmonic series is defined as the sum:

$$h_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}, \quad n \geq 1 \tag{1}$$

The series diverges, but slowly, like $\ln(n)$. A good approximation to h_n is given by the formula:

$$f_n = \gamma + \ln\left(n + \frac{1}{2}\right) \tag{2}$$

where $\gamma = 0.57721\ 56649\dots$ is the so-called Euler-Mascheroni constant. An improved approximation to h_n is given by:

$$g_n = \gamma + \ln\left(n + \frac{1}{2} + \frac{1}{24n}\right) \tag{3}$$

See the Wikipedia articles for more information on these and similar approximations:

[http://en.wikipedia.org/wiki/Harmonic_series_\(mathematics\)](http://en.wikipedia.org/wiki/Harmonic_series_(mathematics))
http://en.wikipedia.org/wiki/Euler-Mascheroni_constant

- a. Use the function **cumsum** to calculate h_n for $n = 1 : 50$. For the same ns , calculate also the values of f_n and g_n using a completely vectorized calculation (no loops). Then, use the function **fprintf** in its vectorized form to generate a table of values formatted exactly as the one shown below:

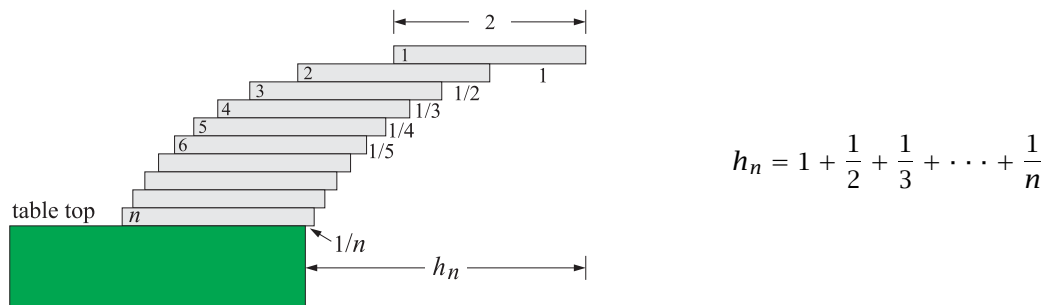
n	h(n)	g(n)	f(n)

1	1.000000	1.010080	0.982681
2	1.500000	1.501805	1.493506
3	1.833333	1.833939	1.829979
4	2.083333	2.083605	2.081293
..... etc.			
48	4.458797	4.458797	4.458779
49	4.479205	4.479206	4.479188
50	4.499205	4.499206	4.499189

You must print all 50 values and use only three **fprintf** commands, two for the headers, and one for the 50×4 matrix of numerical values.

- b. Plot h_n and f_n together on the same graph. Then, plot h_n and g_n . Because both approximations are very good, the curves will fall virtually on top of each other. To discern their differences better, plot the absolute-value differences $|h_n - f_n|$ and $|h_n - g_n|$ on the same graph (you may want to use a **semilogy** type of plot in order to better see both curves.) Annotate your graphs with axis labels, title, and legends.

- c. *Harmonic Stacks*. A somewhat counter-intuitive application of the harmonic series is the problem of stacking identical tiles (or books or dominoes or coins) at the edge of a table with each tile being placed slightly beyond the edge of the tile below it, as shown below. The question is how far can the tiles extend beyond the edge of the table without toppling over.



The problem has been discussed since the 19th century in many Mechanics books, and many variants of it, some very recent, exist. See the zip-file `harmonic-stacks.zip` on sakai (under Resources for Week-2) for a collection of papers on the subject. It turns out that for identical tiles of length a and uniform density, the overhang distance for n tiles is given in terms of the harmonic series h_n by:

$$d_n = \frac{a}{2} h_n = \frac{a}{2} \left(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \right) \quad (4)$$

The above figure shows the case $a = 2$ in which case $d_n = h_n$. Eq. (4) is obtained by requiring that the center of mass of the collection of n tiles be vertically aligned with the edge of the table, and similarly, any subgroup of m tiles counting from the top must have a center of mass aligned at the edge of the $(m + 1)$ -tile, for $m = 1, 2, \dots, n - 1$.

Using the approximation of Eq. (2), determine how many tiles it would take for the overhang distance to be greater than the length of one tile, i.e., the top tile is completely beyond the edge of the table, in other words, find the smallest n such that $d_n \geq a$. *Hint*: Use Eq. (3) to write (4) as

$$d_n \approx \frac{a}{2} \left[\gamma + \ln \left(n + \frac{1}{2} \right) \right] \quad (5)$$

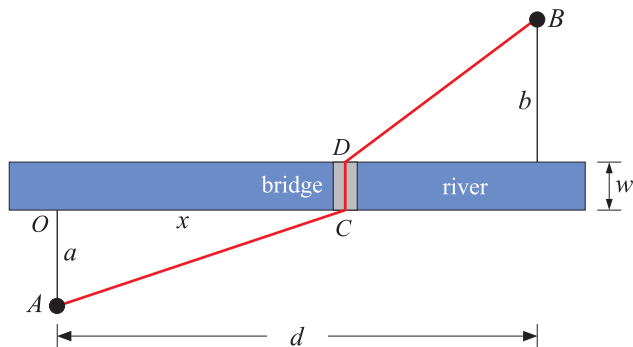
and solve this equation for n in terms of d_n . Repeat by finding the smallest n such that $d_n \geq 2a$, and then $d_n \geq 4a$, and $d_n \geq 8a$. A vectorized calculation that covers all these cases is best, i.e., start by defining the vector $d = [a, 2a, 4a, 8a]$, or $d/a = [1, 2, 4, 8]$.

6. In a joint civil engineering project, two towns A and B decide to build a bridge across a river separating them. The vertical distances of the towns to the river are a, b , the river width is w , and the distance separating the towns along the river is d , as shown below. The total distance (red line) between the two towns through the bridge is the sum of the segments: $L = (AC) + (CD) + (DB)$. The bridge location is defined by the distance x from the point O , that is, $x = (OC)$. The total distance may be thought of as a function of x and is given by:

$$L(x) = \sqrt{x^2 + a^2} + \sqrt{(d - x)^2 + b^2} + w$$

The towns agree to build the bridge at a distance x such that the total travel distance $L(x)$ is *minimized*.

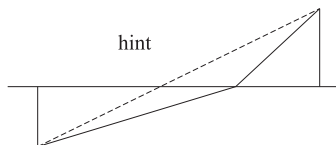
Thus, we wish to find the x that minimizes $L(x)$.



- Consider the particular values $a = 2$, $b = 3$, $d = 10$, $w = 1$ kilometers. Using the function **linspace**, define a vector of 101 equally-spaced values of x in the interval, $0 \leq x \leq d$, and using a vectorized calculation, compute the corresponding 101 values of the total travel distance $L(x)$. Use MATLAB's built-in function **min** to determine the minimum value L_0 of the distance $L(x)$, and the distance x_0 at which it occurs, that is, $L_0 = L(x_0)$. [Hint: see class notes.]
- Plot $L(x)$ versus x and add to the graph the optimum point (x_0, L_0) .
- For arbitrary values of a, b, d, w , it can be shown geometrically (see pictorial hint), or analytically by solving $dL(x)/dx = 0$, that the optimum distance x_0 and minimized travel distance L_0 are:[†]

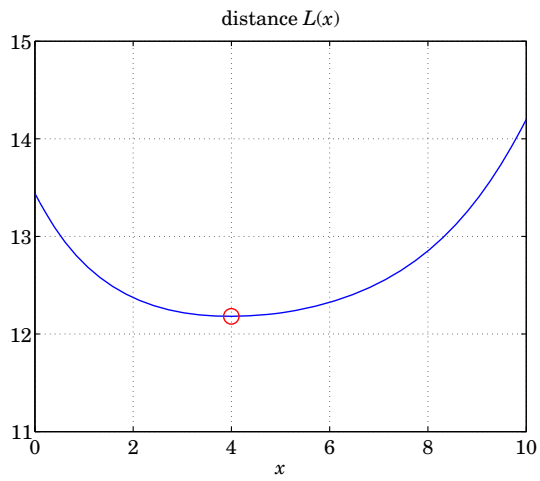
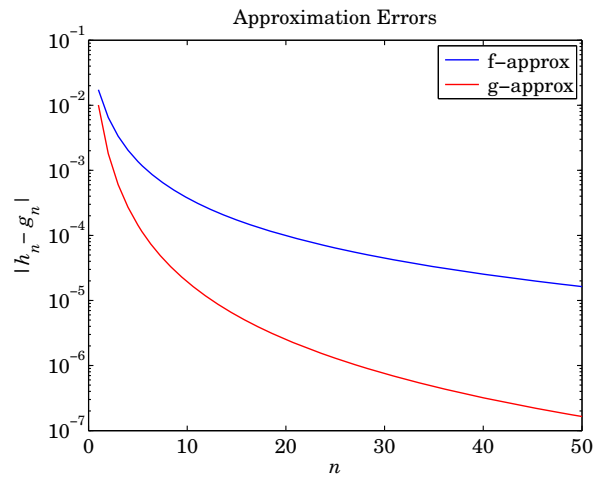
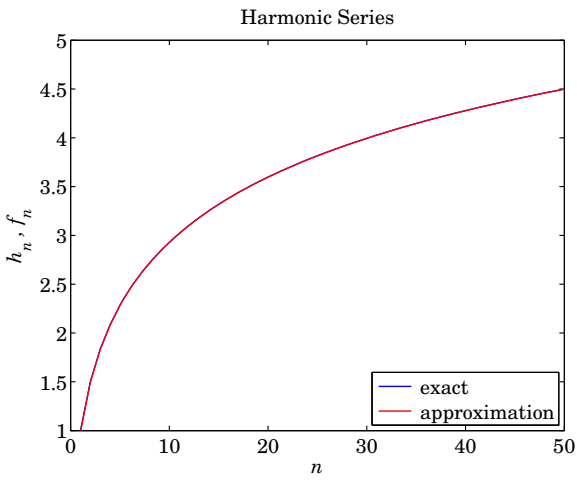
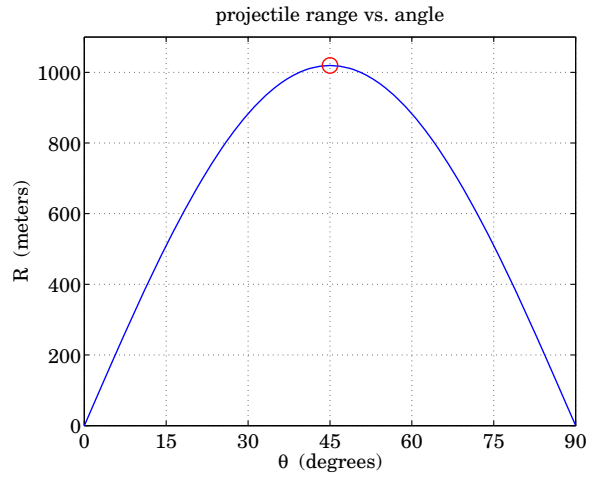
$$x_0 = \frac{ad}{a+b}$$

$$L_0 = w + \sqrt{(a+b)^2 + d^2}$$



- and that x_0 is obtained when the hypotenuses (AC) and (DB) are parallel to each other. Verify that the x_0, L_0 calculated from these formulas agree with those obtained numerically in part (a).
- Determine the minimum point (x_0, L_0) using also the function **fminbnd**, discussed in class.

[†]P. Nahin, *When Least is Best*, Princeton University Press, 2004.



Homework Problems – Week 3
440:127 – Spring 2015 – S. J. Orfanidis

1. End-of-Chapter Problem 4.5. The required data file, `ace_data.dat`, is included. It can be loaded by the following command, which loads it into a 61×5 matrix named, `ace_data`:

```
load ace_data.dat
```

Do part (e) also in the following way: first sort only column 2 of the ACE data matrix in descending order, then, re-arrange the data matrix according to the sorting index of column 2 (see p. 24 of the week-2 lecture notes).

2. End-of-Chapter Problem 4.8. Use step increments of 300 for both T, P and use **meshgrid** to calculate v with a single command. Then, use **fprintf** commands to print your results exactly as shown below, with T listed vertically and P horizontally:

	100	400	700	1000	kPa
100 K	0.29	0.07	0.04	0.03	
400 K	1.15	0.29	0.16	0.11	
700 K	2.01	0.50	0.29	0.20	
1000 K	2.87	0.72	0.41	0.29	

In your solution, you must use only three **fprintf** commands to print the above, two for the first two lines, and a vectorized one for the bottom four lines (no loops).

3. Consider the following function over the interval $0 \leq x \leq 1$:

$$f(x) = x \sin(3\pi x)$$

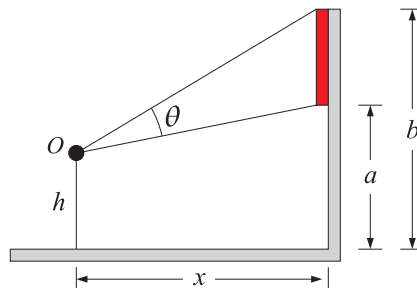
- a. Define an anonymous MATLAB function that implements $f(x)$ and accepts vector inputs and generates vector outputs (see an example on p. 25 of the week-2 lecture notes). Using the function **linspace**, generate an array of 1001 equally-spaced x -values in the interval $0 \leq x \leq 1$ (you may also use the double-colon operator instead of **linspace**), and make a plot of $f(x)$ versus x .
 - b. For the same vector of x s of part (a), use the functions **max** and **min** to determine the locations of the local maxima and local minima of this function over the interval $[0, 1]$ and place them on the graph of part (a). Generate a plot exactly as the one shown at the end of this handout. [Hint: see pp. 32-36 of week-2 lecture notes.]
 - c. Determine more accurate local maxima and minima using the function **fminbnd** and compare them with those found in part (b). Find out how to control the accuracy of the results returned by the function **fminbnd**.
4. *Regiomontanus' problem* (ca. 1470) is considered to be the oldest maximization problem after antiquity. It can be posed as follows: You are viewing a painting hanging high on a museum wall. At what distance x from the wall would your viewing angle θ be maximized?

Let h, a, b , be your height, and the heights of the bottom and top sides of the painting, respectively, as shown below. Assuming $a > h$ and $b > h$, it can be shown by some trigonometry that θ is related to the distance x as follows, with optimum distance x_0 and corresponding maximum angle θ_0 :

$$\theta = \arctan \left[\frac{(b-a)x}{x^2 + (a-h)(b-h)} \right]$$

$$x_0 = \sqrt{(a-h)(b-h)}$$

$$\theta_0 = \arctan \left[\frac{b-a}{2x_0} \right]$$



- a. For the numerical values $a = 9$, $b = 12$ feet, use your own height h (in feet) to calculate your optimum viewing distance x_0 (in feet) and viewing angle θ_0 (in degrees). Calculate x_0, θ_0 for a person that is half your height (so the next time you take your little sister to the museum please be aware that her optimum viewing distance is not the same as yours.)
- b. Set $h = 6$, $a = 9$, $b = 12$ feet. Calculate x_0, θ_0 . Using the function **linspace** define a vector of 100 values of x in the interval $0 \leq x \leq 2x_0$. Using a vectorized calculation (no loops), calculate the corresponding vector of angles θ and plot them versus x . On the same graph, add the single data point (x_0, θ_0) , and confirm that it lies at the maximum of the θ -curve.
- c. Determine estimates of x_0, θ_0 numerically using MATLAB's built-in function **max** and compare them with the exact values. Without recalculation, explain what happens if the array x is taken to be 101 equally-spaced values in the range $[0, 2x_0]$, instead of the above 100 values.
- d. Determine x_0, θ_0 using also the function **fminbnd**. [*Hint*: the maximum of $\theta(x)$ is equivalent to the minimum of its negative, $-\theta(x)$.]

Note: If you would like to prove the above formulas, you may use the following trig identity and inequality:

$$\tan(\theta_1 - \theta_2) = \frac{\tan \theta_1 - \tan \theta_2}{1 + \tan \theta_1 \tan \theta_2} \quad \text{and} \quad \frac{x}{x_0} + \frac{x_0}{x} \geq 2$$

with the inequality being valid for all positive x, x_0 , and with equality reached at $x = x_0$ (this inequality is a consequence of the so-called arithmetic-geometric mean inequality.)

5. It is desired to build a rectangular fence around a 1000 square-foot area. The fence costs one dollar per foot. Let a, b be the sides (in feet) of the rectangular fence so that its area and perimeter are $A = ab$ and $F = 2a + 2b$. Clearly the cost of the entire fence is equal to F dollars.

It is desired to determine a, b so that the total cost F is minimized while the area is kept equal to $A = 1000$. It is straightforward to see that the solution to this problem is when the area is square, so that $a_0 = b_0 = \sqrt{A}$ and $F_0 = 4a_0$. However, here we wish to determine the solution numerically.

Define a vector of side-lengths, $a = 20 : 0.1 : 40$. Then, calculate the corresponding vector of costs F . Using the function **min**, determine the minimum value of the array F , say F_1 , and the value of a where it occurs, say a_1 , as well as the corresponding other side, say b_1 . Compare these values with the exact values a_0, b_0, F_0 .

Moreover, plot F versus a and place on the graph the numerically determined minimum point (a_1, F_1) , as well as the exact minimum (a_0, F_0) , and annotate your graph as shown at the end of this handout.

6. Two formulas for calculating π are:[†]

$$\pi = \frac{9801}{2\sqrt{2}} \left[\sum_{k=0}^{\infty} \frac{(4k)! (1103 + 26390k)}{(k!)^4 (396)^{4k}} \right]^{-1}$$

$$\pi = 2\sqrt{3} \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1) 3^k}$$

The first one is due to Ramanujan[‡] and converges extraordinarily fast. The second is due to Madhava (ca. 1400).^{*} Truncate both series to a maximum number of 10 terms. Define the vector $n = 0 : 9$ and using the built-in functions **cumsum** and **factorial** create a 10×3 matrix of the approximation values, as shown in the table below, where the n th entries are computed by using the terms $0 : n$ only. Your computations must be completely vectorized. For convenience, you may define and work with the following two functions $f(n)$ and $g(n)$ (whose limiting values are π as $n \rightarrow \infty$):

$$f(n) = \frac{9801}{2\sqrt{2}} \left[\sum_{k=0}^n \frac{(4k)! (1103 + 26390k)}{(k!)^4 (396)^{4k}} \right]^{-1}$$

$$g(n) = 2\sqrt{3} \sum_{k=0}^n \frac{(-1)^k}{(2k+1) 3^k}$$

Moreover, use appropriate MATLAB commands, e.g., the function **fprintf**, to print your results as in the table below, where the very last row prints π itself.

n	f(n)	g(n)
0	3.1415927300133055	3.4641016151377544
1	3.1415926535897940	3.0792014356780038
2	3.1415926535897931	3.1561814715699539
3	3.1415926535897931	3.1378528915956805
4	3.1415926535897931	3.1426047456630846
5	3.1415926535897931	3.1413087854628832
6	3.1415926535897931	3.1416743126988376
7	3.1415926535897931	3.1415687159417840
8	3.1415926535897931	3.1415997738115058
9	3.1415926535897931	3.1415905109380797
Inf	3.1415926535897931	

7. A ball thrown vertically upwards with initial velocity v in a uniform gravitational field with acceleration of gravity g reaches a height h given by:

$$h = \frac{v^2}{2g}$$

In a variation of Example 4.3 of the textbook, consider the values of g given in Table 4.2 for various planetary objects in our solar system. Define a row vector of these values, $g = [3.7, 8.87, \dots, 0.58]$. It is desired to calculate the heights reached on each planet when the ball is thrown with the three initial velocities $v = [5, 10, 15]$ m/sec.

Use **meshgrid** to calculate all such heights simultaneously (in units of meters) and use **fprintf** to print your results in a table exactly as shown below. [*Hint*: Define the planet names as a cell array of strings.

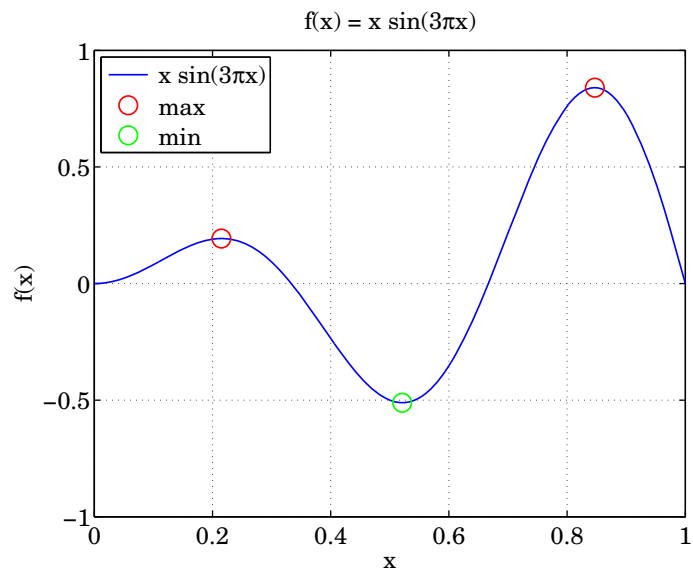
[†]<http://en.wikipedia.org/wiki/Pi>

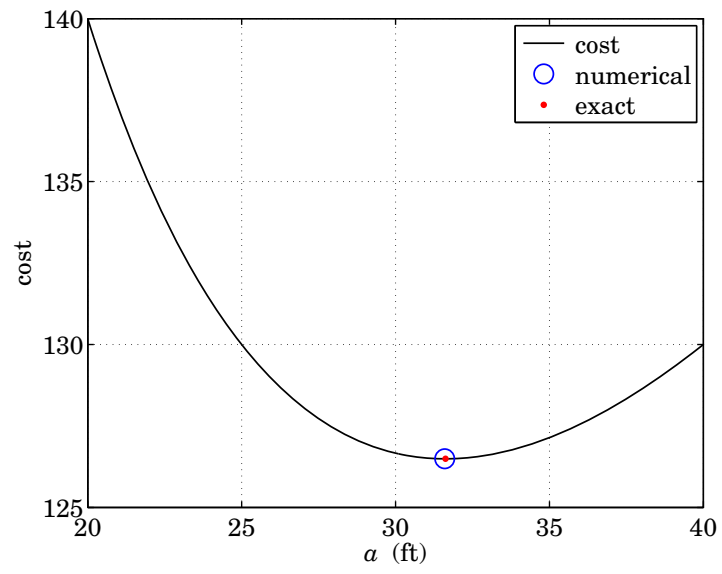
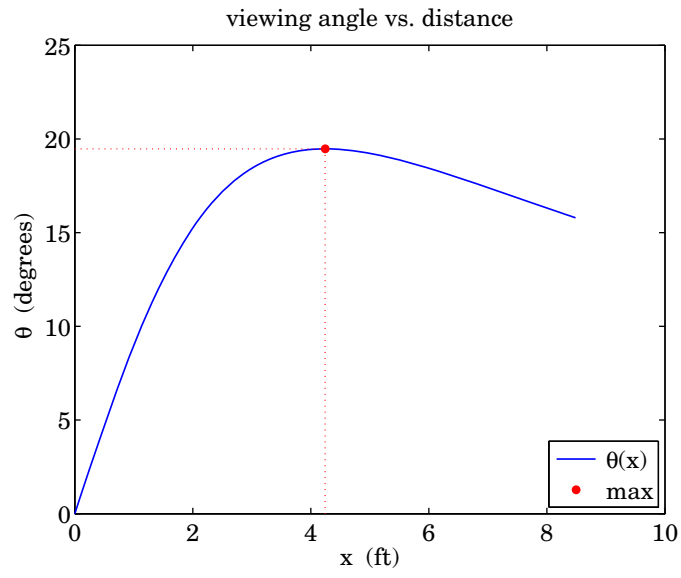
[‡]<http://en.wikipedia.org/wiki/Ramanujan>

^{*}http://en.wikipedia.org/wiki/Madhava_of_Sangamagrama

Look at Example 4 on p. 60 of the week-1 lecture notes on how to print both text and numerical columns using cell array definitions, and how to left-justify the strings.]

	v =	5	10	15
Mercury	g = 3.70	3.38	13.51	30.41
Venus	g = 8.87	1.41	5.64	12.68
Earth	g = 9.80	1.28	5.10	11.48
Moon	g = 1.60	7.81	31.25	70.31
Mars	g = 3.70	3.38	13.51	30.41
Jupiter	g = 23.12	0.54	2.16	4.87
Saturn	g = 8.96	1.40	5.58	12.56
Uranus	g = 8.69	1.44	5.75	12.95
Neptune	g = 11.00	1.14	4.55	10.23
Pluto	g = 0.58	21.55	86.21	193.97





Homework Problems – Week 4
440:127 – Spring 2015 – S. J. Orfanidis

1. End-of-Chapter Problem 5.12.
2. End-of-Chapter Problem 5.17. Plot parts (a,b,c,d) in separate figure windows, not as subplots.
3. End-of-Chapter Problem 5.20.
4. End-of-Chapter Problem 5.31.
5. This is a continuation of Problem 5.17. When you get a loan or mortgage payable in a fixed number of years, the payment amount (per payment period) can be calculated from the formula:

$$x = \frac{ry_0(1+r)^N}{(1+r)^N - 1} \quad (1)$$

where y_0 is the loan amount and, assuming monthly payments, $r = R/12$, where R is the annual interest rate (e.g., for a 6% annual rate, $R = 0.06$ and $r = R/12 = 0.005$), and N is the total number of payment periods (e.g., $N = 12 \times 5 = 60$ months for a 5-year loan). Let y_n denote the loan balance at the end of the n -th payment period. At the next period, $n + 1$, your payment of x dollars is subtracted from the present balance of y_n , but before this is done, the bank charges you interest $i_n = ry_n$, therefore, only the amount $b_n = x - ry_n$ is used to reduce your balance, so that the next balance is:

$$y_{n+1} = y_n - b_n = y_n - (x - ry_n) = (1+r)y_n - x$$

This recursion can be solved analytically, and after using Eq. (1), we obtain the solution:

$$y_n = \frac{(1+r)^N - (1+r)^n}{(1+r)^N - 1} y_0, \quad n = 0, 1, 2, \dots, N \quad (2)$$

The solution correctly gives $y_n = y_0$ at $n = 0$, and $y_n = 0$ at $n = N$. Consider the numerical values $y_0 = 10000$, $R = 0.06$, $N = 60$.

- a. Compute the payment amount x and the values of the balance y_n for $n = 0, 1, \dots, N$, as well as the interest i_n paid at the n -th period, and the amount b_n used to reduce the balance, and using **fprintf** make a table exactly like the one shown below (you must print all 61 entries).

n	yn	bn	in

0	10000.00	143.33	50.00
1	9856.67	144.04	49.28
2	9712.63	144.76	48.56
3	9567.86	145.49	47.84
4	9422.37	146.22	47.11

etc. -----			

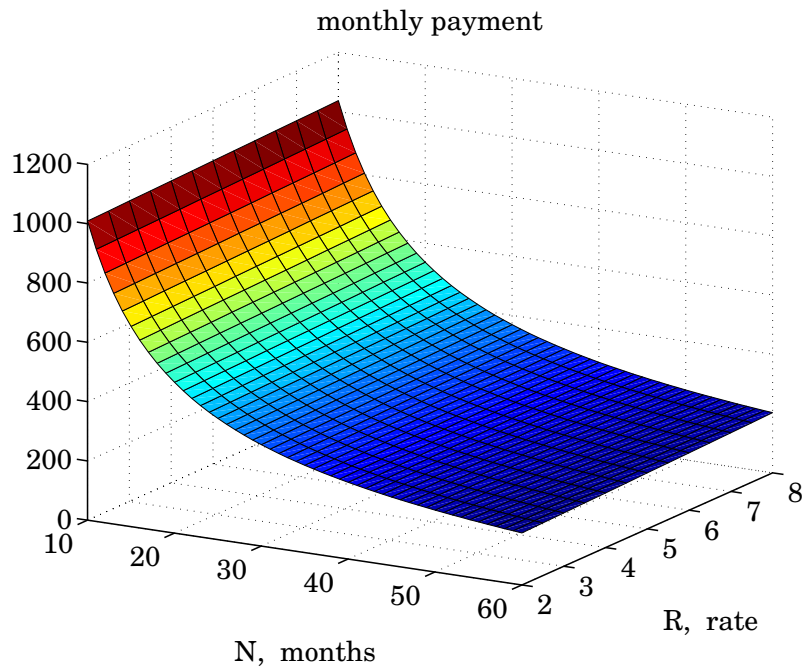
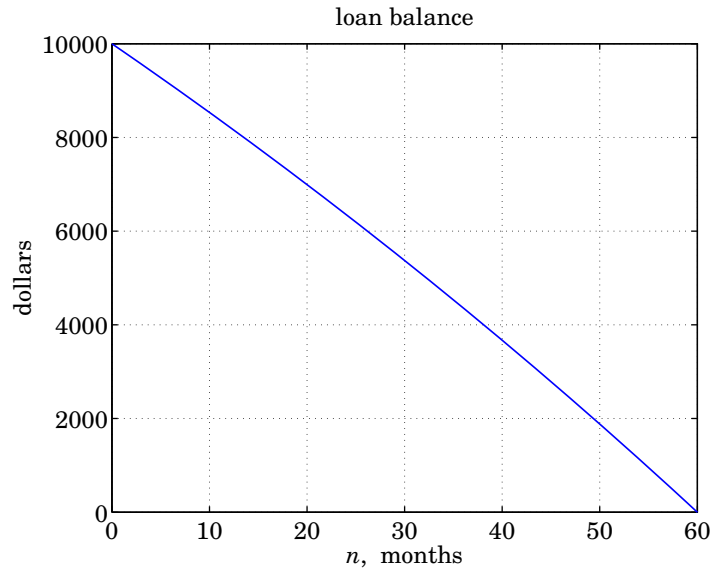
56	763.74	189.51	3.82
57	574.23	190.46	2.87
58	383.78	191.41	1.92
59	192.37	192.37	0.96
60	0.00	-	-

Note that initially the interest payment is high but it gets smaller with time, while the amount that goes to repay the loan increases. Banks usually give you a payment schedule just like this table.

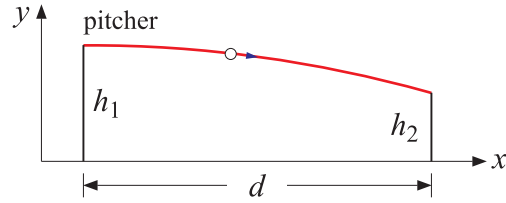
- b. Using the function **sum** compute the total amount that went to re-pay the loan (i.e., the sum of the b_n s), and the total amount of interest paid.

Make a plot of y_n versus n and observe how it is driven to zero at $n = N$.

- c. The payment amount x given in Eq. (1) can be thought of as function of two independent variables N, R . Using **meshgrid** and **surf**, make a surface plot like the one shown below by assuming that N and R range over $10 \leq N \leq 60$ months, and $2 \leq R \leq 8$ annual percentage rates.



6. A pitcher throws a ball to a batter located at a distance d , with the ball being launched from height h_1 with velocity v_0 and angle θ_0 , and arriving t seconds later at the batter's position at height h_2 . The ball's x - y trajectory (neglecting air resistance and curved balls) is described by,



$$\begin{aligned} x &= v_0 t \cos \theta_0 \\ y &= h_1 + v_0 t \sin \theta_0 - \frac{1}{2} g t^2 \end{aligned} \quad (3)$$

where g is the acceleration of gravity. In this problem, we wish to plot the ball's trajectory, and moreover, calculate and plot the travel time and pitch angle as functions of the pitcher's velocity v_0 .

Substituting $t = x / (v_0 \cos \theta_0)$ and using the trigonometric identity, $1 + \tan^2 \theta_0 = 1 / \cos^2 \theta_0$, we may re-write the y -equation in the following form, which shows that the trajectory is parabolic:

$$y = h_1 + x \tan \theta_0 - \frac{g x^2}{2 v_0^2} (1 + \tan^2 \theta_0) \quad (4)$$

Given the pitcher's velocity v_0 and setting $x = d$ and $y = h_2$, we obtain two conditions from which one can calculate the required launch angle θ_0 and travel time t in terms of v_0 :

$$h_2 = h_1 + d \tan \theta_0 - \frac{g d^2}{2 v_0^2} (1 + \tan^2 \theta_0), \quad t = \frac{d}{v_0 \cos \theta_0} \quad (5)$$

The first equation is quadratic in the variable $\tan \theta_0$ and can be solved using the quadratic formula. It has two solutions which are both valid, but the one that has the smaller θ_0 leads to a faster travel time. In order for the solutions to exist as real numbers (i.e. the ball can reach the batter), it is necessary that v_0 be larger than a certain minimum value, which can be determined by setting the discriminant of the quadratic equation (5) to zero. This gives the minimum velocity:

$$v_{0,\min} = \sqrt{-gh + g\sqrt{h^2 + d^2}}, \quad h = h_1 - h_2 \quad (6)$$

- a. Let $h_1 = 5$, $h_2 = 2$, $d = 60$ feet and $g = 32$ ft/sec². First, calculate the minimum velocity (6) in miles per hour[†]. Then, for the pitcher velocity of $v_0 = 50$ miles/hr, calculate the two possible launch angles θ_0 in degrees and the corresponding travel times t in seconds from Eqs. (5).

For the two found angles θ_0 make a plot of the trajectories in Eq. (4) versus x in the range $0 \leq x \leq d$, placing both trajectories on the same graph.

- b. Next choose a vector of equally-spaced velocity values in the interval, $40 \leq v_0 \leq 100$ mph, and calculate the corresponding vectors of angles θ_0 in degrees (using the fast solution), and travel times t in seconds.

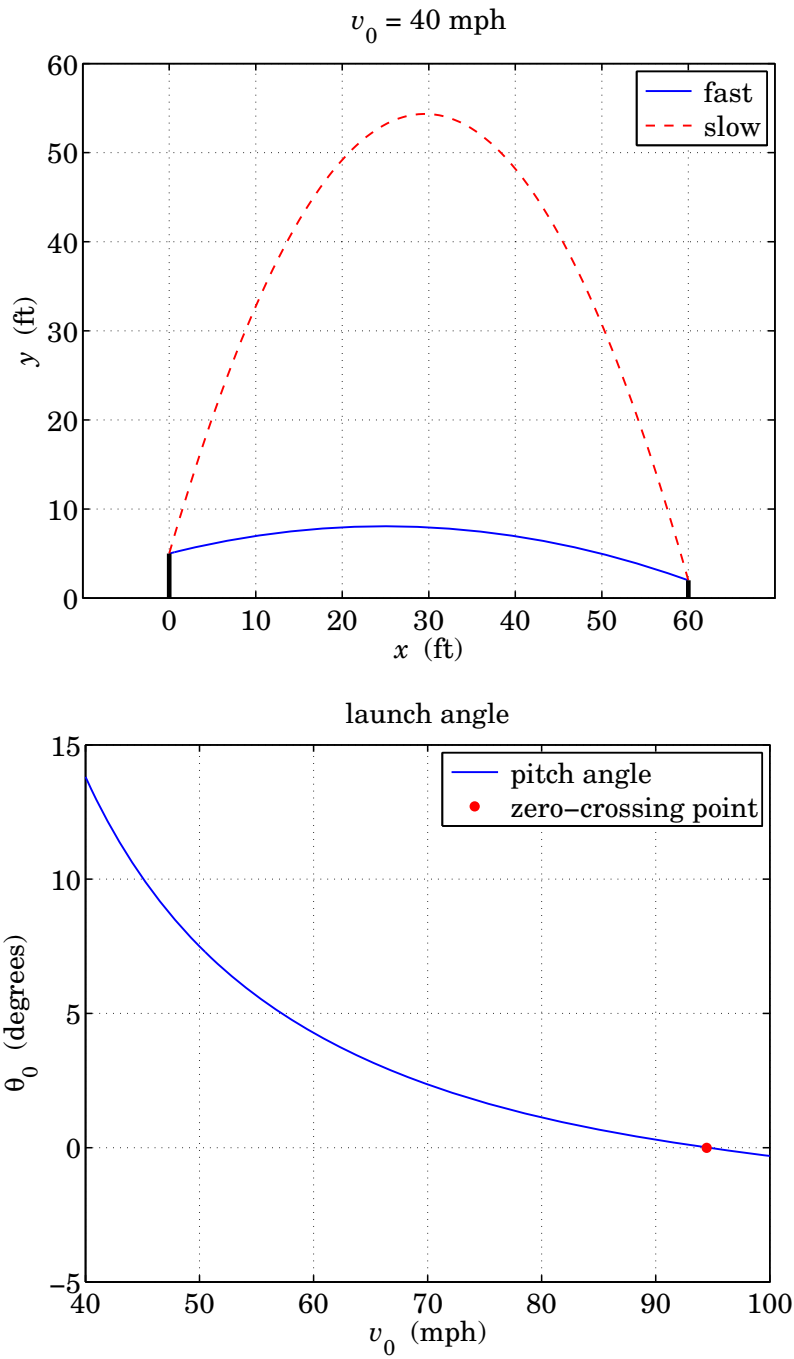
Plot θ_0 versus v_0 , then, plot t versus v_0 . In both cases, use units of mph for v_0 in the plots.

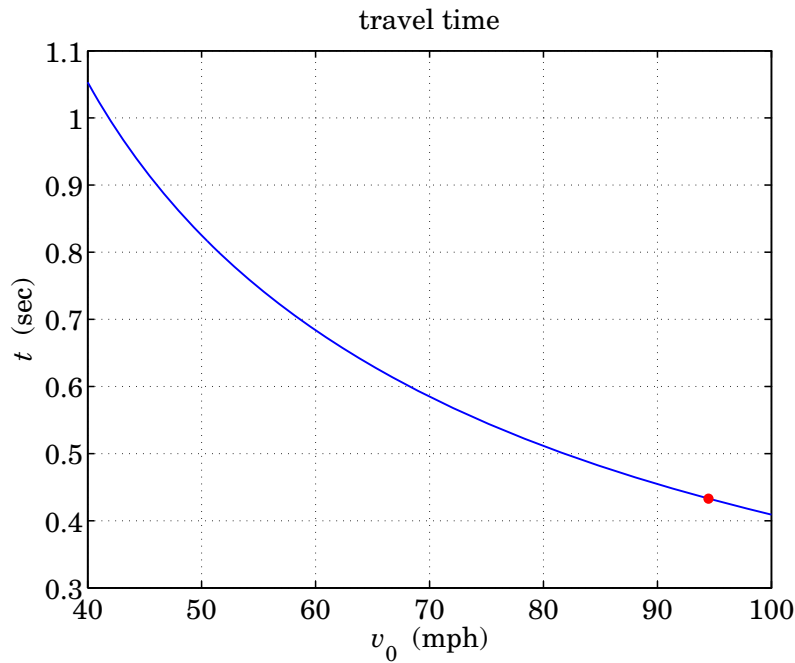
Determine the speed v_z in mph at which the pitch angle goes from positive to negative, and place that point on the angle graph. Calculate also the corresponding travel time for this speed and place it on the time graph.

- c. For a fast pitcher (e.g., $v_0 > 95$ mph), because the distance $d = 60$ feet is fairly short, the effect of gravity will be minimal. For the same velocity vector v_0 of part (b), calculate and plot the travel times in the case when gravity is completely ignored, but on the same graph place the travel times of part (b) for comparison.

[†] 1 mile = 5280 feet

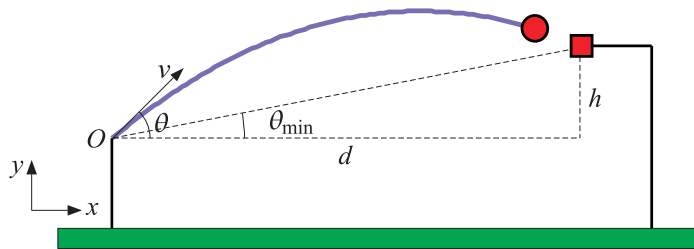
Hints: $\theta_0 = \arctan\left(\frac{v_0^2 \pm \sqrt{v_0^4 + 2v_0^2gh - g^2d^2}}{gd}\right)$, $v_z = \sqrt{\frac{gd^2}{2h}}$





7. This is a continuation of the hoops movie example and explains the math and physics contained in the file hoops.m. Anyone who has shot a basketball knows that the shooting speed and shooting angle must be interrelated in order to reach the hoop. Moreover, most people try to shoot at the lowest possible speed, because it makes it easier to aim.

At time t during the flight, the x, y coordinates of the moving basketball are given by the typical projectile equations, shown below in Eq. (7), where v, θ are the initial launch speed and angle and g is the acceleration of gravity, $g = 9.81 \text{ m/s}^2$.



$$\begin{aligned} x &= v \cos \theta t \\ y &= x \tan \theta - \frac{1}{2} \frac{g x^2}{v^2 \cos^2 \theta} \end{aligned} \quad (7)$$

The condition to reach the hoop is that $y = h$ when $x = d$, which gives the speed-angle relationship:

$$d = h \tan \theta - \frac{1}{2} \frac{g d^2}{v^2 \cos^2 \theta} \Rightarrow v = \sqrt{\frac{g d / 2}{\cos^2 \theta (\tan \theta - h / d)}} \quad (8)$$

The corresponding time of flight to reach the hoop is given by,

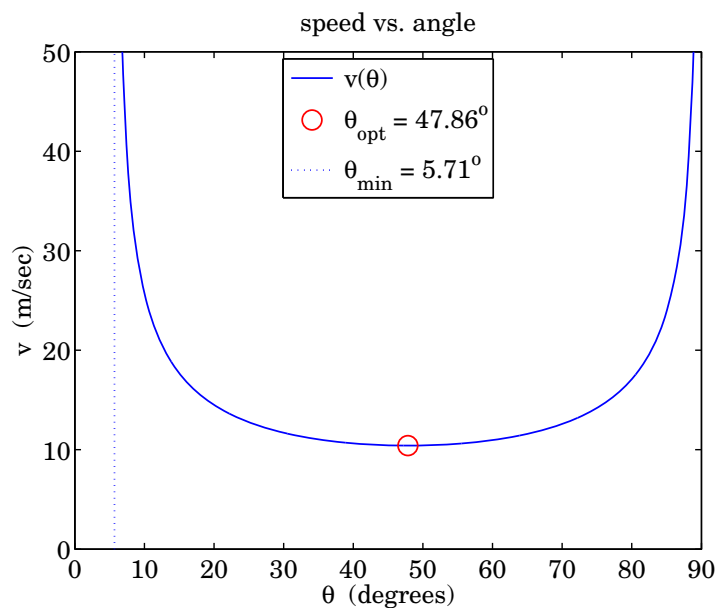
$$t = \frac{d}{v \cos \theta} = \sqrt{\frac{2d}{g} (\tan \theta - h / d)} \quad (9)$$

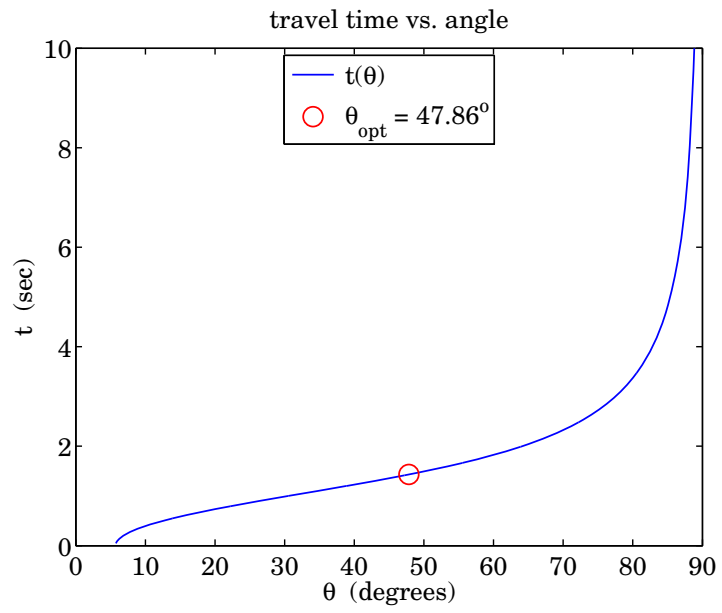
It is clear from Eq. (8) that for v to be real-valued, the angle θ must satisfy the restriction $\tan \theta \geq h / d$, or, $\theta \geq \theta_{\min} = \arctan(h / d)$. The geometrical meaning of θ_{\min} is shown in the above figure. Clearly, if the basketball is shot below the line connecting the launch point and the hoop, the ball will fly below the hoop, no matter the initial speed. Note also that $v = \infty$ at $\theta = \theta_{\min}$ and at $\theta = 90^\circ$. Therefore, there is

a minimum value of v somewhere in the range, $\theta_{\min} < \theta < 90^\circ$. From calculus, the minimum of v can be determined by differentiating Eq. (8) with respect to θ and setting the derivative to zero. This leads to the optimum values for θ and v ,

$$\theta_{\text{opt}} = \arctan \left[\frac{h}{d} + \sqrt{1 + \frac{h^2}{d^2}} \right], \quad v_{\text{opt}} = \sqrt{g d \tan \theta_{\text{opt}}} \quad (10)$$

- Set $h = 1$ and $d = 10$ meters. Define anonymous functions $v(\theta)$ and $t(\theta)$ for Eq. (8) and (9). Plot $v(\theta)$ over the range $\theta_{\min} < \theta < 90^\circ$. To avoid infinities, do the plot over the more restricted range $1.01 \theta_{\min} \leq \theta \leq 89.9^\circ$. [You need to exercise some care in converting the angles between radians and degrees]. Place on the graph the optimum point $\theta_{\text{opt}}, v_{\text{opt}}$. Note how broad the minimum is, which explains the fact that there is typically quite a leeway in choosing a low shooting speed and corresponding angle.
- Plot $t(\theta)$ versus θ over the same range as in part (a), and add the point that corresponds to θ_{opt} .
- Determine the optimum point $\theta_{\text{opt}}, v_{\text{opt}}$ in two other ways, (i) using the **min** function, and (ii) using the **fminbnd** function, and compare them with the exact values given in Eq. (10).





Homework Problems – Week 5
440:127 – Spring 2015 – S. J. Orfanidis

1. This problem is a combination of Problems 6.7 and 6.14 of the text. A rocket is launched vertically. At time $t = 0$, the rocket's engine shuts down. At that time, the rocket has reached an altitude of $h_0 = 500$ meters and is rising at a velocity of $v_0 = 125$ m/sec. Gravity then takes over. The height of the rocket as a function of time is:

$$h(t) = h_0 + v_0 t - \frac{1}{2} g t^2, \quad t \geq 0$$

where $g = 9.81$ m/s² is the acceleration of gravity. The time $t = 0$ marks the time the engine shuts off. After this time, the rocket continues to rise and reaches a maximum height of h_{\max} meters at time $t = t_{\max}$. Then, it begins to drop and reaches the ground at time $t = t_g$.

- a. Create a function called **height** that accepts time as an input and returns the height of the rocket. The quantities h_0, v_0 should be additional inputs and the function should have syntax:

$$h = \text{height}(t, h_0, v_0);$$

where t is a vector of times and h stands for the corresponding vector of heights $h(t)$. Your function can be either an M-file, or defined anonymously. Use your function in your solutions to parts (b–e).

- b. Make a preliminary plot of the height versus time for times from 0 to 28 seconds. Use an increment of 0.5 seconds in your time vector.
- c. Find the maximum height reached h_{\max} and the time t_{\max} when the rocket starts to fall back to the ground in two ways (i) using the **max** function, and (ii) using the **fminbnd** function and your function **height**.
- d. Using the function **fzero**, find the time t_g when the rocket hits the ground (i.e., when the function value is zero).
- e. Evaluate and plot the rocket height $h(t)$ at 301 equally-spaced points in the time interval $0 \leq t \leq t_g$, and add to the graph the maximum point (t_{\max}, h_{\max}) , as well as the point when the rocket hits the ground at $t = t_g$.

Some things to consider:

The quantities h_{\max}, t_{\max}, t_g can be determined analytically from physics, but in this problem we wish to calculate them numerically using **fminbnd** and **fzero**. For your reference, the exact formulas are:

$$t_{\max} = \frac{v_0}{g}, \quad h_{\max} = h_0 + \frac{v_0^2}{2g}, \quad t_g = t_{\max} + \sqrt{t_{\max}^2 + \frac{2h_0}{g}}$$

In applying **fminbnd**, you need to specify a search interval. You may choose that to be $[0, 2v_0/g]$ since it includes the maximum. Similarly, in applying **fzero**, you need to specify a nearby search point, which you can also choose it to be $2v_0/g$.

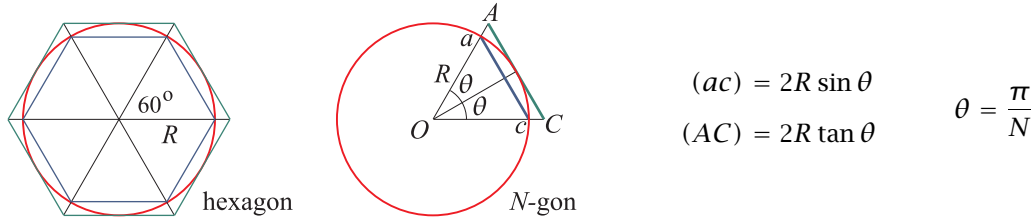
2. *Archimedes' Algorithm for π* . In a very short, remarkable, work entitled *Measurement of a Circle*, Archimedes proved 23 centuries ago three fundamental geometrical facts: (i) that the area of a circle is given by πR^2 , a formula every school child knows, (ii) that the circumference of a circle ($2\pi R$) involves the same constant π that appears in the area formula, and (iii) he gave a systematic algorithm for calculating π to any desired degree of accuracy. This algorithm remained the preferred method for calculating π for the next 2000 years. In this homework, we will derive Archimedes' algorithm and implement it in MATLAB. Some variations of the algorithm, such as Liu Hui's, and other even more efficient modern algorithms, such as Ramanujan's, will be studied in other homework sets. Currently, π has been calculated to trillions of digits. Note that the letter π was introduced in this context by William Jones in the 1700s and was popularized by Euler. You may find the following references useful:

W. Dunham, *Journey Through Genius*, Wiley, New York, 1990.

<http://math.nyu.edu/~crrorres/Archimedes/contents.html>

<http://en.wikipedia.org/wiki/Pi>

Archimedes' method was to bracket the value of π between a lower and an upper bound that can be made to converge to π . The method is illustrated below. The circumference of a circle is approximated from below by the circumference of an inscribed N -sided polygon, and from above by the circumference of the corresponding circumscribed polygon. The figure shows the case of a hexagon ($N = 6$). Starting with the hexagon, Archimedes then kept dividing the subtending angle by half, considering polygons of ever increasing number of sides, given by $N = 6 \cdot 2^n$, $n = 0, 1, 2, \dots$. For an N -gon, the subtending angle of each of the N inscribed triangles is $2\theta = 2\pi/N$, or $\theta = \pi/N$.



By multiplying the base sides (ac) and (AC) shown above by N , we obtain the total circumferences of the inscribed and circumscribed polygons. These bracket the circle circumference, i.e.,

$$N \cdot (ac) < 2\pi R < N \cdot (AC) \Rightarrow 2NR \sin \theta < 2\pi R < 2NR \tan \theta$$

Canceling out a factor of $2R$ and replacing $\theta = \pi/N$, we obtain the basic algorithm:

$$N \sin \left(\frac{\pi}{N} \right) < \pi < N \tan \left(\frac{\pi}{N} \right), \quad N \geq 3 \quad (1)$$

By the way, Liu Hui's algorithm (265 CE), provides a tighter upper bound and is given by:

$$N \sin \left(\frac{\pi}{N} \right) < \pi < N \sin \left(\frac{\pi}{N} \right) \left[2 - \cos \left(\frac{\pi}{N} \right) \right], \quad N \geq 3$$

Setting $N = N_0 2^n$ in Eq. (1), where N_0 is the starting N_0 -gon (e.g., $N_0 = 6$), we obtain Archimedes' algorithm:

$$N_0 2^n \sin \left(\frac{\pi}{N_0 2^n} \right) < \pi < N_0 2^n \tan \left(\frac{\pi}{N_0 2^n} \right), \quad n = 0, 1, 2, \dots \quad (2)$$

You may justifiably ask what good are these formulas if both the upper and lower bounds involve π , which is what we wish to calculate. However, these bounds can be calculated geometrically, as Archimedes did, or via a recursive procedure that we will study in a later homework. Here, we will have MATLAB evaluate these expressions directly for various values of N .

- a. Let $a(N), b(N)$ denote the left and right bounds in Eq. (1) viewed as functions of N . Write a MATLAB function called **bounds** with syntax:

```
[a,b] = bounds(N);
```

that computes the functions $a(N), b(N)$ at any vector of N s. Next, define the following row vector N whose values are of historical significance:

$$N = [6, 6 \cdot 2^4, 360, 6 \cdot 2^9, 6 \cdot 2^{11}, 6 \cdot 2^{16}] = [6, 96, 360, 3072, 12288, 393216] \quad (3)$$

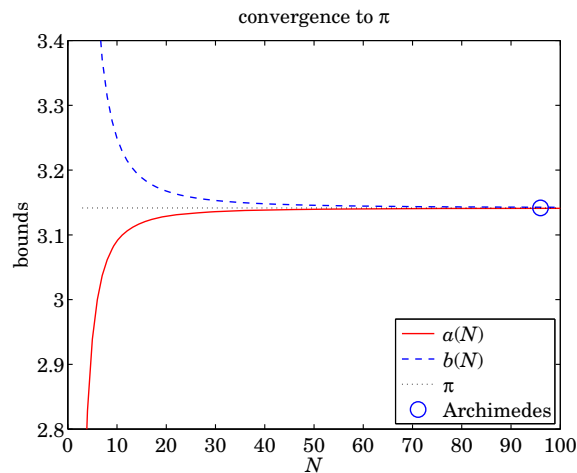
Evaluate $a(N), b(N)$ at these values of N and use `fprintf` to generate the following table of values using appropriate MATLAB commands:

N	a(N)	b(N)	
6	3.0000000000	3.46410161514	Hexagon
96	3.14103195089	3.14271459965	Archimedes (230 BCE)
360	3.14155277941	3.14167240467	Ptolemy (150)
3072	3.14159210600	3.14159374877	Liu Hui (265)
12288	3.14159261937	3.14159272204	Zu Chongzi (480)
393216	3.14159265356	3.14159265366	Viète (1590)

In particular, note that the last two cases compute π to 7- and 10-digit accuracy. For comparison, you may print the true value of π with 10-digit accuracy using:

```
vpa(pi,11)
ans =
3.1415926536
```

- b. Next, define the vector $N = [3, 4, 5, \dots, 100]$ and, on the same graph, plot the functions $a(N), b(N)$ versus N , and add the horizontal line at π and indicate the point corresponding to Archimedes' computation at $N = 96$. In particular, figure out how to use MATLAB commands to generate your graph in the following form:



- c. Write another MATLAB function, **bounds2**, with syntax:

```
[a,b] = bounds2(n,N0);
```

that implements the left and right bounds of Eq. (2), where now the independent variable n is the exponent in the polygon's dimension $N = N_0 2^n$. Use the value $N_0 = 6$. The functions must accept a vector of n s and generate the corresponding vectors of function values $a(n), b(n)$.

For the values $n = [0, 1, 2, \dots, 16]$, plot $a(n), b(n)$ versus n , and generate a graph like the above, indicating Archimedes' point at $n = 4$.

- d. Write a MATLAB function, **polygon**, with syntax:

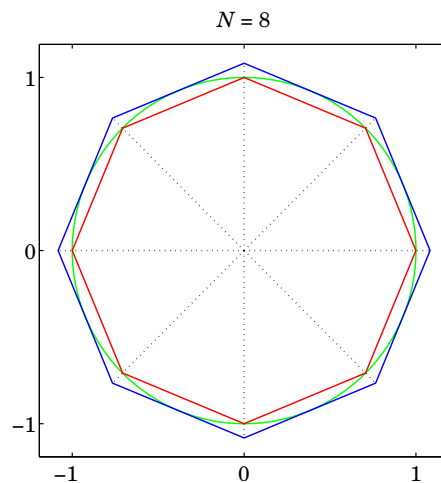
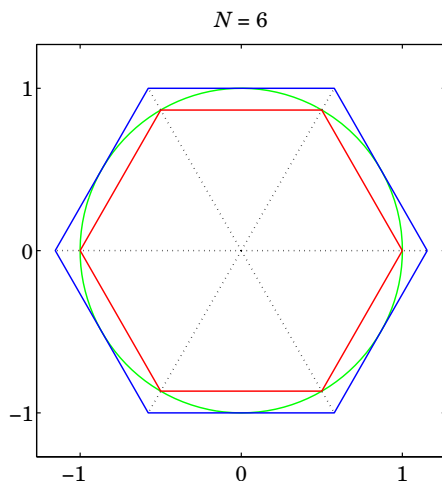
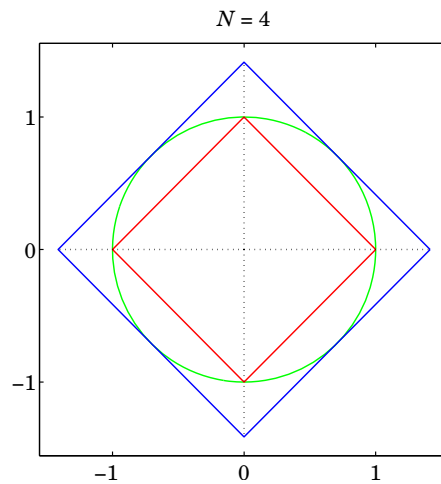
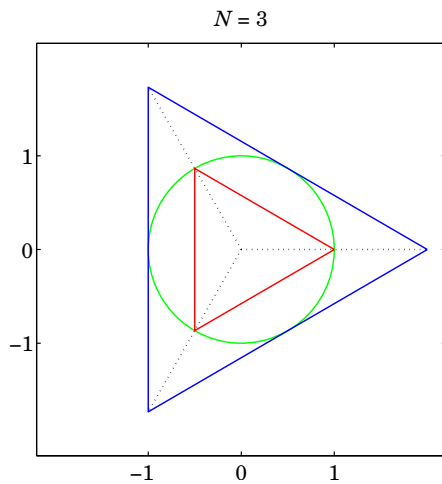
```
polygon(N);
```

that generates a plot of a circle and an inscribed and a circumscribed N -sided regular polygon. The function's input must be a scalar N , and its output must be a graph like those at the end of the handout, including the title and color schemes. To get started, the following code segment is given that you might build upon:

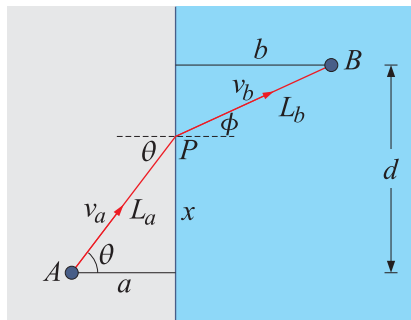
```
th = 0: 2*pi/N : 2*pi;  
x = cos(th); y = sin(th);  
figure; plot(x,y,'r-');
```

```
axis equal; axis square;  
title(['\itN} = ',num2str(N)]); % we'll discuss num2str() later
```

Using this function, generate the polygon graphs for the cases $N = 6, 12, 96$.



3. As part of a triathlon event, an athlete has the task of running from a fixed point A on the beach to some point P on the shoreline, and then swimming from P to a fixed point B , as shown below. Her ground and swimming speeds are v_a, v_b , respectively. To win this part of the race, she must determine the point P , or equivalently, the optimum bearing angle θ that she should maintain on the ground, in order to minimize the total travel time from A to B .[†] The total travel time can be expressed as a function of the bearing angle θ by Eq. (4) below.



$$T(\theta) = \frac{L_a(\theta)}{v_a} + \frac{L_b(\theta)}{v_b} \quad (4)$$

where L_a, L_b are the lengths of the segments $(AP), (PB)$.

- Assuming that the parameters a, b, d, v_a, v_b are given, write a function $T(\theta)$ using a one-line anonymous definition that implements Eq. (4), where θ must be entered in degrees.
- Clearly, the optimum angle θ will be somewhere in the interval $0 \leq \theta \leq \theta_{\max}$, where θ_{\max} is defined through $d = a \tan \theta_{\max}$. Consider the numerical values (in metric units):

$$a = 300, \quad b = 400, \quad d = 500, \quad v_a = 3, \quad v_b = 2$$

Calculate θ_{\max} in degrees. Then, using the function **fminbnd**, determine the optimum bearing angle θ_0 in degrees that minimizes $T(\theta)$, and the minimized value $T_0 = T(\theta_0)$ in minutes.

- Create a vector of 300 angles θ in the range $0 \leq \theta \leq \theta_{\max}$. Plot $T(\theta)$ versus θ , with $T(\theta)$ expressed in minutes, and add to the graph the optimum point θ_0, T_0 .
- If you've had a course in Optics, you may recognize this as a problem of refraction at an interface between two media. Snell's law of refraction relates the optimum angles θ, ϕ by

$$\frac{\sin \theta}{\sin \phi} = \frac{v_a}{v_b}$$

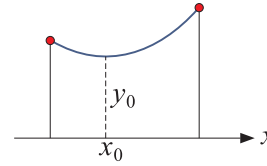
Verify with MATLAB that this relationship is indeed satisfied for the computed optimum angles.

Historical note: In the refraction context, the above method of minimizing the time of travel across the media, leading to Snell's law, is a special case of "Fermat's Principle of Least Time". It was introduced by Fermat in 1660, and is used for ray tracing of light through inhomogeneous media. It applies to all types of waves, light, sound, seismic, etc.

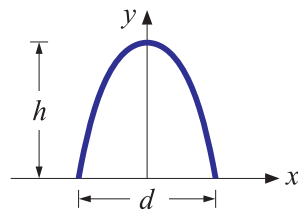
[†]R. P. Feynman, et al., *Feynman Lectures on Physics*, vol. I, Pearson-Addison-Wesley, San Francisco, 1963 and 2006. A. Gilat, *MATLAB, An Introduction with Applications*, Wiley, New York, 2011.

4. The *catenary* is the shape of a chain hanging under its own weight.[†] It can be found all around us: hanging chains and ropes, power lines, telephone lines, necklaces and bracelets, simple suspension bridges (not ones supporting a roadway), tents, mooring ropes and anchor chains of ships, catenary risers in offshore drilling platforms, minimal surfaces of revolution such as soap films. The catenary curve is a scaled hyperbolic cosine:

$$y = y_0 + a \cosh\left(\frac{x - x_0}{a}\right) - a$$



where a, x_0, y_0 are parameters that are fixed by the end-point conditions. It was realized 350 years ago by Robert Hooke that an *inverted catenary* should be the optimal shape for the construction of arches, in the sense that the net force (gravity plus tension) exerted at each part of the arch should be directed tangentially along the arch (you will learn more about that in your Statics courses). The equation describing such an optimal arch is:



$$y = h - a \cosh\left(\frac{x}{a}\right) + a, \quad -\frac{d}{2} \leq x \leq \frac{d}{2} \quad (5)$$

where d, h are the base and height, respectively, and we assumed that the arch is placed symmetrically about the origin. A famous example is the St. Louis *Gateway Arch*, which is not quite an inverted catenary, but approximates one closely.[‡] Given the values of d, h , the arch shape parameter a is fixed by requiring that $y = 0$ at the end-points, $x = \pm d/2$. This gives the nonlinear equation for a :

$$a \cdot \cosh\left(\frac{d}{2a}\right) - a = h \quad (6)$$

- Using the built-in MATLAB function **fzero**, find the parameter a for the values $d = 20, h = 10$ feet. You may use the crude approximation $a_0 = d^2/(4h)$ as the initial estimate of a (this is derived by replacing $\cosh(x)$ by its Taylor series approximation, $\cosh(x) = 1 + x^2/4 + \dots$)
- Plot the arch function in Eq. (5) over the interval $-d/2 \leq x \leq d/2$.
- The outer perimeter of the Gateway Arch has dimensions $d = h = 630$ feet. Repeat parts (a,b) for this case.
- The mathematical equations that describe the Gateway Arch are given as follows for the centroid curve of the arch, that is, the curve that passes through the center of mass of each triangular cross section of the arch, where all distances are in feet (for details, see Osserman's article):

$$\begin{aligned} y &= h - A \cosh(Bx) + A \\ A &= 68.7672, \quad B = 0.0100333 \\ d &= 598.4478, \quad h = A \cosh(Bd/2) - A = 625.0772 \end{aligned} \quad (7)$$

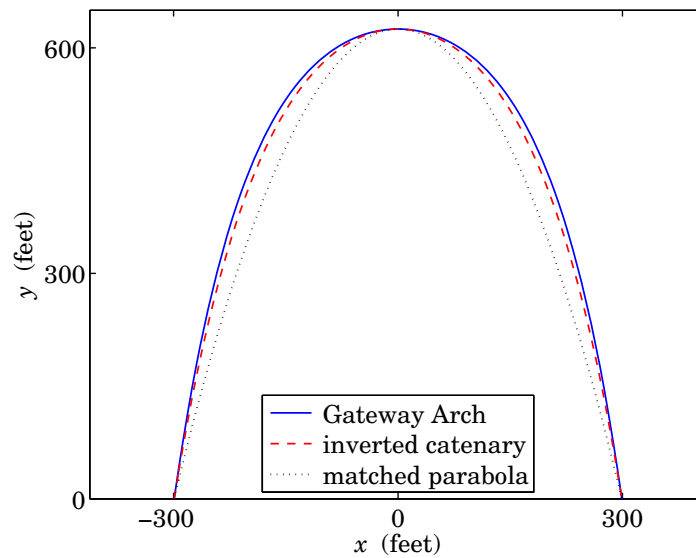
For these values of d, h , design an inverted catenary and determine its shape parameter a from Eq. (6). In addition, determine the parameters C, D of a parabola that matches the base and height d, h of the arch (i.e. express C, D in terms of d, h):

$$y = C + Dx^2 \quad (8)$$

[†]<http://en.wikipedia.org/wiki/Catenary>

[‡]See R. Osserman's interesting article (on Sakai) discussing the history and mathematics of the Gateway Arch.

Then, evaluate Eqs. (7) and (8), as well as Eq. (5), at $-d/2 \leq x \leq d/2$, and plot all three curves on the same graph. Observe how closely the catenary approximates the true arch.



Note: For small values of the ratio h/d , i.e., less than 0.2, the catenary and matched parabola are virtually indistinguishable. In fact, Galileo thought that the mathematical form of a hanging chain was a parabola. For a suspension bridge that supports a roadway, such as the Verrazano or the Golden Gate Bridge, it can be shown that the shape of the supporting cables is actually parabolic (note that the Golden Gate has $h/d = 0.18$.)

If you are connected to the internet, you can load an image of the Gateway Arch from Wikipedia into MATLAB as discussed in the week-4 lecture notes.



Homework Problems – Week 6
440:127 – Spring 2015 – S. J. Orfanidis

1. The attached file, `set6a.dat`, has the following contents:

X	A	B	Y	Z
-10.005	bb	A	300.00005	200.00
15.025	dddd	CCC	-6.12300	40000.00
6.705	a	DDDD	-130.10009	10.00
8.002	ccc	EEEE	70.50000	3000.00

- Open this file using `fopen`, skip over the two header lines using `fgetl`, and use a single `fscanf` command to read the three numerical columns into a 4×3 matrix. From that matrix, extract and display the individual columns X, Y, Z . Rewind but do not close the file.
- Then, use a single `textscan` command to read from the file the two text columns A, B , each being a 4×1 cell array of strings. Display A, B on the screen, for example, B should display as:

```
'A'  
'CCC'  
'DDDD'  
'EEEE'
```

- Using the `sort` function, sort column Y in ascending order and then sort the other columns X, A, B, Z according to Y 's sorting index. Using `fopen`, open a new data file for writing called, `set6b.dat`, and using `fprintf`, write into it the sorted columns X, A, B, Y, Z , including the original header lines. The sorted file should have contents like these:

X	A	B	Y	Z
6.705	a	DDDD	-130.10009	10.00
15.025	dddd	CCC	-6.12300	40000.00
8.002	ccc	EEEE	70.50000	3000.00
-10.005	bb	A	300.00005	200.00

Use the following `type` command to display the contents of the new file on the screen:

```
type set6b.dat
```

2. The included data file, `set6c.dat`, contains water level data for Lake Powell for the years 2000–2007. Please see Problem 8.8 of the text for the context of this problem. The data file is essentially the same as Table 8.9 of the text and its contents are:

Lake Powell Water-Level Data (source: Problem 8.8 of the textbook)

	2000	2001	2002	2003	2004	2005	2006	2007
January	3680.12	3668.05	3654.25	3617.61	3594.38	3563.41	3596.26	3601.41
February	3678.48	3665.02	3651.01	3613.00	3589.11	3560.35	3591.94	3598.63
March	3677.23	3663.35	3648.63	3608.95	3584.49	3557.42	3589.22	3597.85
April	3676.44	3662.56	3646.79	3605.92	3583.02	3557.52	3589.94	3599.75
May	3676.76	3665.27	3644.88	3606.11	3584.70	3571.60	3598.27	3604.68
June	3682.19	3672.19	3642.98	3615.39	3587.01	3598.06	3609.36	3610.94
July	3682.86	3671.37	3637.53	3613.64	3583.07	3607.73	3608.79	3609.47
August	3681.12	3667.81	3630.83	3607.32	3575.85	3604.96	3604.93	3605.56
September	3678.70	3665.45	3627.10	3604.11	3571.07	3602.20	3602.08	3602.27
October	3676.96	3663.47	3625.59	3602.92	3570.70	3602.31	3606.12	3601.27
November	3674.93	3661.25	3623.98	3601.24	3569.69	3602.65	3607.46	3599.71
December	3671.59	3658.07	3621.65	3598.82	3565.73	3600.14	3604.96	3596.79

- a. Open the data file with the function `fopen`, and after skipping over the four header lines, use a single `fscanf` command to read the numerical columns into a 12×8 matrix. Using the function `mean`, compute and print the yearly average levels (the means of the columns), and the monthly averages (the means of the rows), as well as the overall mean (for all 12×8 months).
- b. Open a new data file for writing called, `set06d.dat`, and using appropriate `fprintf` commands, write the water level data into this file, and append a last column that contains the monthly averages, and a bottom row that contains the yearly averages, as well as the overall average at the bottom-right corner. The file contents should look exactly as follows:

Lake Powell Water-Level Data (source: Problem 8.8 of the textbook)

	2000	2001	2002	2003	2004	2005	2006	2007	monthly-ave
January	3680.12	3668.05	3654.25	3617.61	3594.38	3563.41	3596.26	3601.41	3621.94
February	3678.48	3665.02	3651.01	3613.00	3589.11	3560.35	3591.94	3598.63	3618.44
March	3677.23	3663.35	3648.63	3608.95	3584.49	3557.42	3589.22	3597.85	3615.89
April	3676.44	3662.56	3646.79	3605.92	3583.02	3557.52	3589.94	3599.75	3615.24
May	3676.76	3665.27	3644.88	3606.11	3584.70	3571.60	3598.27	3604.68	3619.03
June	3682.19	3672.19	3642.98	3615.39	3587.01	3598.06	3609.36	3610.94	3627.27
July	3682.86	3671.37	3637.53	3613.64	3583.07	3607.73	3608.79	3609.47	3626.81
August	3681.12	3667.81	3630.83	3607.32	3575.85	3604.96	3604.93	3605.56	3622.30
September	3678.70	3665.45	3627.10	3604.11	3571.07	3602.20	3602.08	3602.27	3619.12
October	3676.96	3663.47	3625.59	3602.92	3570.70	3602.31	3606.12	3601.27	3618.67
November	3674.93	3661.25	3623.98	3601.24	3569.69	3602.65	3607.46	3599.71	3617.61
December	3671.59	3658.07	3621.65	3598.82	3565.73	3600.14	3604.96	3596.79	3614.72
yearly_ave	3678.11	3665.32	3637.93	3607.92	3579.90	3585.70	3600.78	3602.36	3619.75

Use the following command to display the contents of the new file on the screen:

```
type set6d.dat
```

Your `fprintf` commands should not contain any actual numerical values, but rather they should act on the arrays obtained in part (a). See p. 28 the the week-6 lecture notes for an example on how to use a for-loop to print text and numerical columns, as well as the header lines.

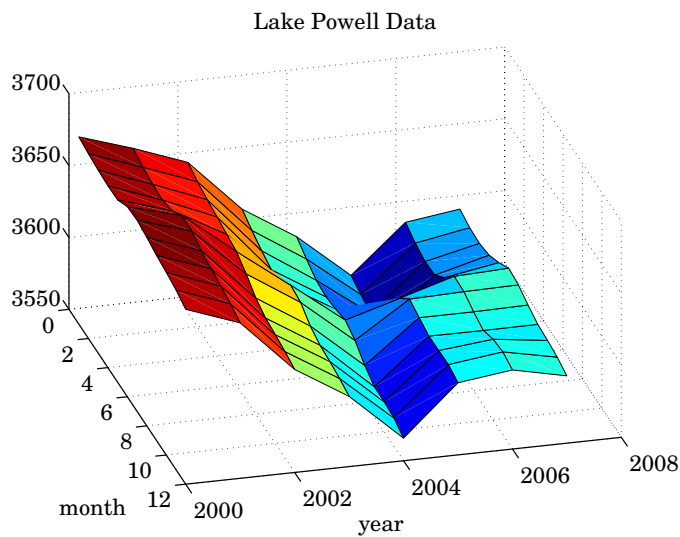
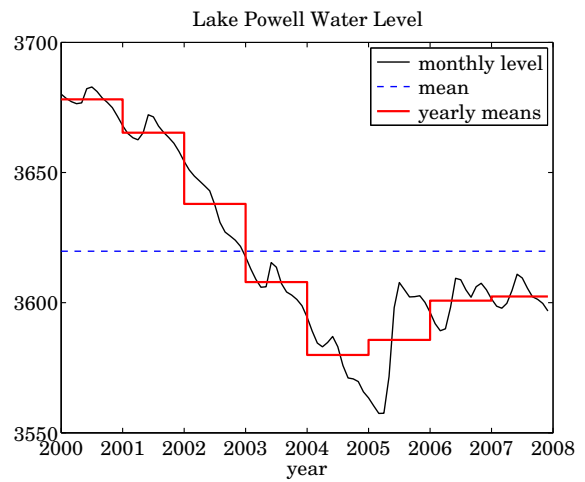
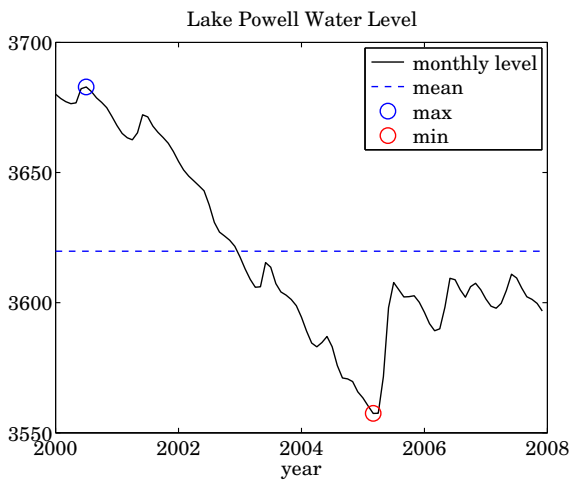
- c. Concatenate all columns of the numerical matrix of part (a) into a column vector of length $12 \times 8 = 96$ that represents consecutive months in consecutive years. Plot it versus time over $2000 \leq t < 2008$. [Hint: Go in monthly steps of $1/12$ of a year, note that the last month in the data is $2008 - 1/12$.] Add to the graph the overall mean level. Using the functions `max` and `min`, determine the maximum and minimum water levels and the time they occur and add them to the graph as open circles. Moreover, determine (using MATLAB commands) the month and year when these maxima and minima occur. Finally, prepare a graph like the one at the end.

d. To better see the average levels, make a separate plot of the concatenated monthly levels and add to it the yearly averages (each extending over a period of 12 months). You may use the plotting function **stairs** for this part. Generate a graph like the one below.

To understand how to use the **stairs** command, try and compare the following code examples:

```
x=0:7; y=8*x-x.^2; [x;y]
figure; stairs(x, y); xaxis(0, 9, 0:9); grid on
figure; stairs([x,8], [y,nan]); xaxis(0, 9, 0:9); grid on
figure; stairs([x,8], [y,0]); xaxis(0, 9, 0:9); grid on
```

e. Finally, using **meshgrid** and **surf**, do a surface plot of the data matrix extracted in part (a), like the one shown below.



3. The file `NFL.dat` contains the stats for some well-known quarterbacks from the 1990s. They are listed alphabetically in the form:

C	T	I	Y	Rating	Player
59.3073	4.40223	3.57542	7.33810	?	Ken Anderson
57.0970	6.38867	4.89174	7.67469	?	Len Dawson
58.4635	3.97135	3.25521	7.15299	?	Tony Eason
..... etc.					
59.6949	5.25424	4.47458	7.44373	?	Danny White
54.5700	5.59198	4.87852	7.75916	?	Johnny Unitas

where the columns labeled C, T, I, Y contain the following data:

C = percent completions, T = percent touchdowns
 I = percent interceptions, Y = average yards gained per pass attempt

The NFL rates the passing ability of quarterbacks by combining these columns with certain weights. Although the NFL does not disclose the details of how they do this, it is possible to reverse-engineer their published ratings and arrive at the following ratings formula (we will derive this in a later homework):

$$R = \frac{50 + 20C + 80T - 100I + 100Y}{24} \quad (1)$$

The objective of this problem is to read the NFL data file, compute the ratings for each player, place them in the column with the question-marks, sort the file from the highest-rated player to the lowest, and save the sorted data in another file.

- Place the data file in the current working folder, or somewhere in MATLAB's path. Open the file with the editor and observe that there are nine header lines above the data, then close the editor. Open the file with `fopen`, skip over the first nine header lines using `fgetl`, then using a single `fscanf` command read the numerical columns into a 20×4 matrix, then rewind the file, skip over the header lines again (but save them this time into cell arrays), and using a single `textscan` command read the last two columns of first and last names into two cell arrays [*Hint*: see the week-6 lecture notes on how to do all these operations.]
- Using Eq. (1), calculate the column vector R of ratings for the players. Then, sort the ratings vector R in decreasing order, saving both the sorted ratings vector and its sorting order. From that sorting order, do a similar sorting of the C, T, I, Y columns as well as the cell arrays of the first and last names.
- Open a new data file, say, `NFLs.dat`, and save into it the sorted data, including the sorted ratings column which will replace the question marks, and include also the nine header lines. Close the file and use the following command to print its contents on the screen:

```
type NFLs.dat
```

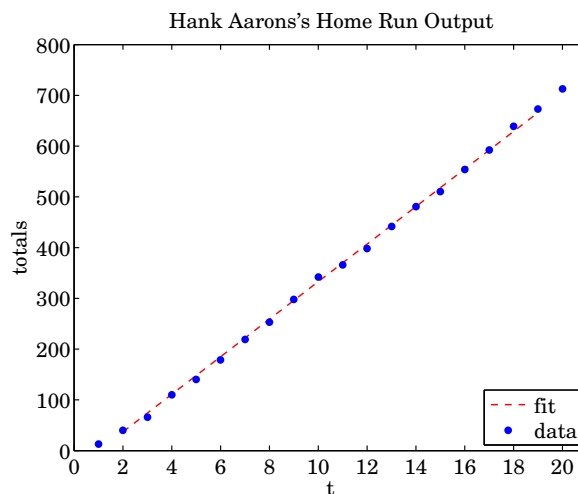
The sorted file should look something like this (not all header lines are shown):

C	T	I	Y	Rating	Player
63.8827	5.32151	2.63612	7.65065	93.951	Joe Montana
59.5616	6.02740	3.42466	7.63096	89.336	Dan Marino
..... etc.					
56.0564	4.86084	3.95923	7.13054	78.214	Bert Jones
54.5700	5.59198	4.87852	7.75916	78.201	Johnny Unitas

4. The file `aaron.dat` contains Hank Aaron's home run output for the 20-year period 1954–1973. Except for the header lines, the file consists of numerical columns.
 - a. Skipping over the header lines with `fgetl`, read the numerical columns into a matrix A , using a single `fscanf` command.
 - b. Let t and H denote the second and third columns of A representing the number of years in the majors and the home runs in that year. Calculate the **cumsum** of H , which represents the total number of home runs up to a given year, and denote that by C .
 - c. In another homework, we will learn how to fit a straight line to this data. The following straight line $y = 37x - 37$ provides an adequate fit to the data from year 2 to year 19. Make a plot of C versus t using dot-markers only (do not connect them by straight line segments), and add to the graph the above fitted straight line. Produce a graph like that shown below.
 - d. Using `fprintf` commands, save the computed cumulative data C together with the original data in a new file, say, `aaron2.dat`, that looks exactly like `aaron.dat`, but has the column labeled "total career" filled with the column C . Display the file on the screen as follows:

Hank Aaron's Home Run Output

calendar year	years in majors	home runs	total career
1954	1	13	13
1955	2	27	40
1956	3	26	66
1957	4	44	110
1958	5	30	140
1959	6	39	179
1960	7	40	219
1961	8	34	253
1962	9	45	298
1963	10	44	342
1964	11	24	366
1965	12	32	398
1966	13	44	442
1967	14	39	481
1968	15	29	510
1969	16	44	554
1970	17	38	592
1971	18	47	639
1972	19	34	673
1973	20	40	713



5. The file NYCtemp.dat contains the following temperature data for New York for the years 1971-75:

source: The Weather Almanac, 5th ed., J. A. Ruffner and F. E. Bair, eds.,
Gale Research Co., Book Tower, Detroit, MI, 1987

year	jan	feb	mar	apr	may	jun	jul	aug	sep	oct	nov	dec
1971	27.0	35.1	40.1	50.8	61.4	74.2	77.8	75.9	71.6	62.7	45.1	40.8
1972	35.1	31.4	39.8	50.1	63.3	67.9	77.2	75.6	69.5	53.5	44.4	38.5
1973	35.5	32.5	46.4	53.4	59.5	73.4	77.4	77.6	69.5	60.2	48.3	39.0
1974	35.3	31.7	42.1	55.2	61.0	69.0	77.2	76.4	66.7	54.1	48.2	39.4
1975	37.3	35.8	40.2	47.9	65.8	70.5	75.8	74.4	64.2	59.2	52.3	35.9

a. Our first task is to read the data file, rearrange it so that the months run vertically and the years horizontally, and save it in a new file, say, NYCtemp2.dat. The new file should preserve the header lines and look as follows:

source: The Weather Almanac, 5th ed., J. A. Ruffner and F. E. Bair, eds.,
Gale Research Co., Book Tower, Detroit, MI, 1987

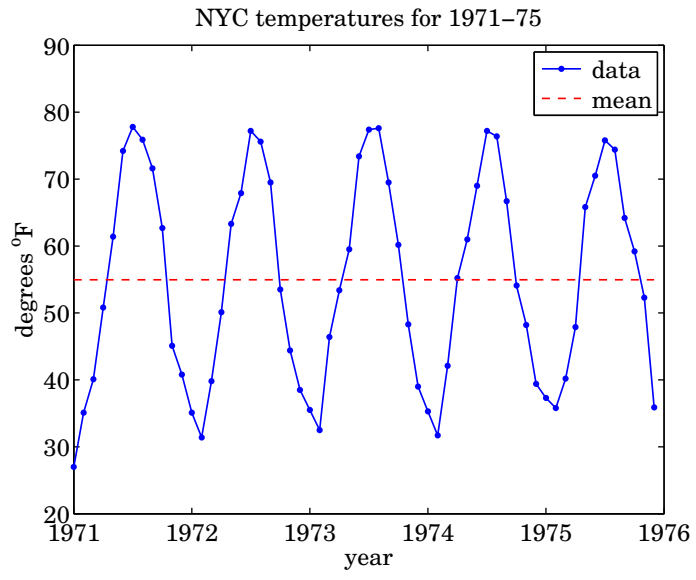
	1971	1972	1973	1974	1975
January	27.0	35.1	35.5	35.3	37.3
February	35.1	31.4	32.5	31.7	35.8
March	40.1	39.8	46.4	42.1	40.2
April	50.8	50.1	53.4	55.2	47.9
May	61.4	63.3	59.5	61.0	65.8
June	74.2	67.9	73.4	69.0	70.5
July	77.8	77.2	77.4	77.2	75.8
August	75.9	75.6	77.6	76.4	74.4
September	71.6	69.5	69.5	66.7	64.2
October	62.7	53.5	60.2	54.1	59.2
November	45.1	44.4	48.3	48.2	52.3
December	40.8	38.5	39.0	39.4	35.9

b. The second task is to find all the months and corresponding year that had temperatures in the nice range of $65 \leq T \leq 75$ °F, and present them in a table, such as the following:

the nicest months & years were:

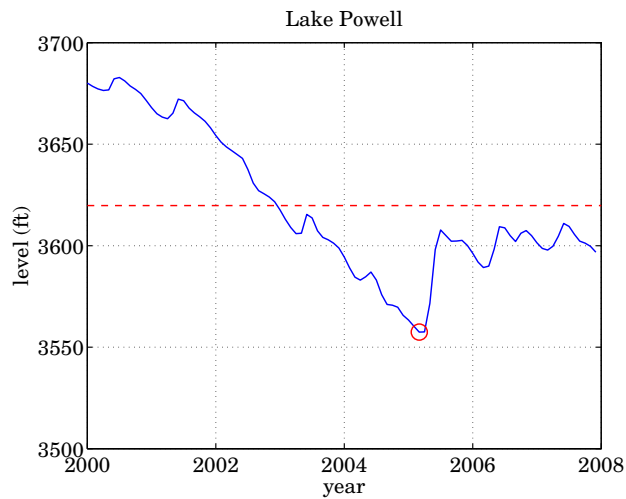
June	1971	T = 74.2 F
September	1971	T = 71.6 F
June	1972	T = 67.9 F
September	1972	T = 69.5 F
June	1973	T = 73.4 F
September	1973	T = 69.5 F
June	1974	T = 69.0 F
September	1974	T = 66.7 F
May	1975	T = 65.8 F
June	1975	T = 70.5 F
August	1975	T = 74.4 F

c. The third task is to plot the above temperature data versus time and indicate the overall average temperature on the graph, as shown below.

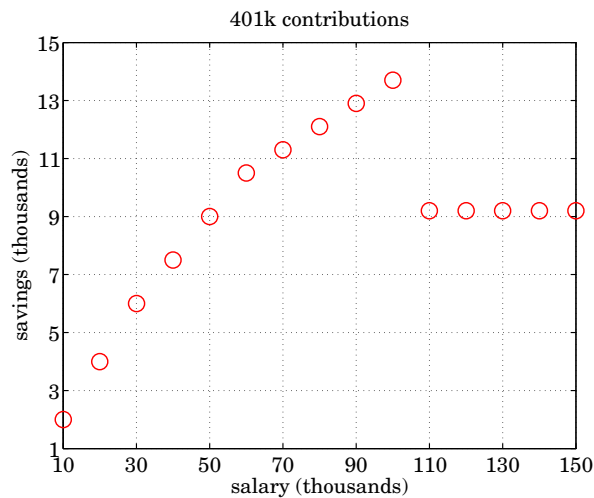


Homework Problems – Week 7
440:127 – Spring 2015 – S. J. Orfanidis

1. End-of-Chapter Problem 8.3.
2. End-of-Chapter Problem 8.7. Data file `temp.dat` is attached.
3. End-of-Chapter Problem 8.10. Data file `lake_powell.dat` is attached. See an example graph below. In addition to Parts (a–d), please do the following: (e) plot the water level versus time over the 8-year period, using tickmarks at 2000 : 2 : 2008, and draw a horizontal line at the overall mean water level. (f) Using `min`, find the month and year when the reservoir was at its lowest and place that point on the previous graph, (g) Using `fprintf` generate a table that looks exactly like Table 8.9 of the text.



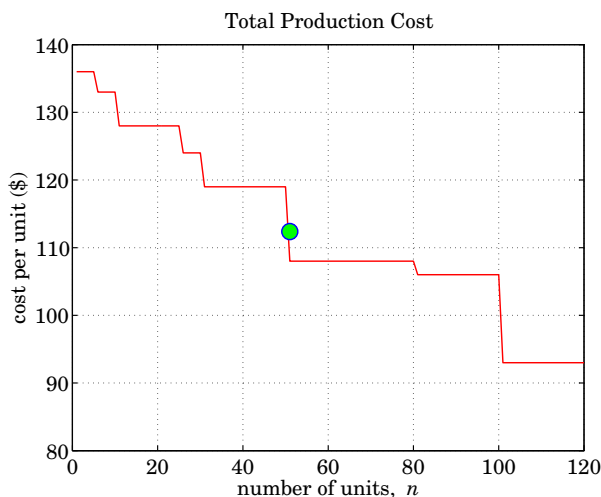
4. End-of-Chapter Problem 8.16.
 After you construct this function, use it to determine the total contribution amounts for the following salaries: \$ 25,000, \$ 50,000, \$ 80,000, and \$ 120,000, and use `fprintf` to make a table of the results. Moreover, use your function to generate a plot like the one shown below.



5. *Production Costs.*^{†‡} A manufacturing company produces a product that has three components A, B, C . The table below shows the cost per unit of the three components as a function of the number n of units produced. The first two columns M, S represent the manufacturing and shipping costs per unit.

n	M	n	S	n	A	n	B	n	C
1-50	\$40.00	1-10	\$20.00	1-5	\$30.00	1-10	\$26.00	1-30	\$20.00
51-100	\$35.00	11-30	\$18.00	6-25	\$27.00	11-50	\$23.00	31-100	\$18.00
101+	\$30.00	31-50	\$15.00	26-100	\$23.00	51-80	\$20.00	101+	\$15.00
		51+	\$12.00	101+	\$18.00	81+	\$18.00		

- Write a function $T(n)$ that computes the total production cost per unit as a function of the number of units n (i.e., the total cost of the columns M, S, A, B, C). Then, for $n = 1 : 120$, make a plot of $T(n)$ versus n , and verify that the more units produced, the cheaper the total cost.
- Moreover, find the average production cost, say T_{av} per unit, and the number of units n_{av} beyond which the total cost drops below the average, and place the point n_{av}, T_{av} on the above graph.



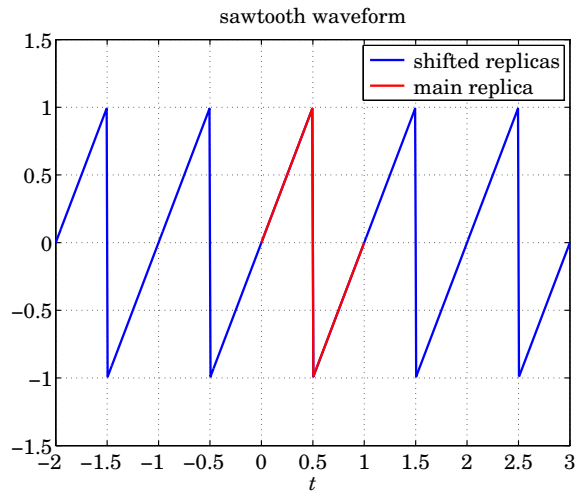
6. Consider a periodic sawtooth waveform defined over one period $0 \leq t \leq 1$ by:

$$f(t) = \begin{cases} 2t, & \text{if } 0 \leq t < 0.5 \\ 0, & \text{if } t = 0.5 \\ 2t - 2, & \text{if } 0.5 < t < 1 \\ 0, & \text{for all other } t \end{cases}$$

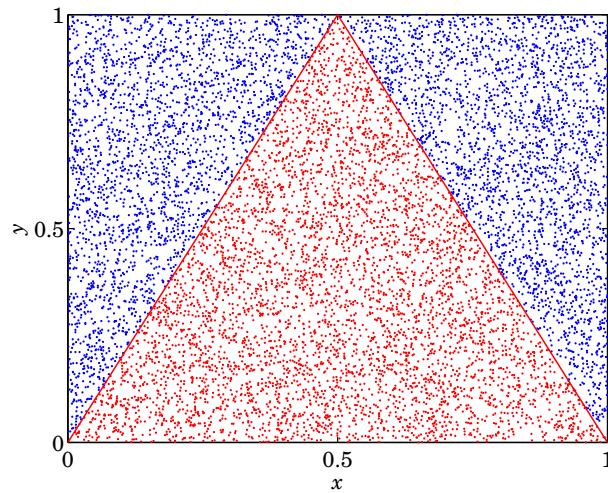
- Write a one-line fully vectorized anonymous MATLAB function $f(t)$ that evaluates the above function at any vector of times t . Plot this function over the interval $0 \leq t \leq 1$.
- Using this function write MATLAB code to reproduce the following plot of $f(t)$, periodically replicated over the interval $-2 \leq t \leq 3$, as shown below.

[†] D. Morrell, *Freshman Engineering Problem Solving with MATLAB*, <http://cnx.org/content/co110325/1.18>, PDF on sakai.

[‡] <http://cnx.org/content/m13433/1.6>



7. We wish to carry out a Monte Carlo calculation of the area of a triangle of base and height equal to one, inscribed inside a rectangle of sides equal to one, as shown below.



Following the discussion of a similar example in week-4 lecture notes, generate $N = 10^4$ random (x, y) pairs that are uniformly distributed inside the rectangle.

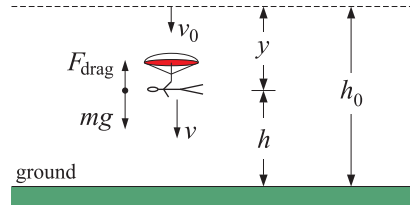
Using appropriate relational operators and the function **find**, determine those (x, y) pairs that lie inside the triangle, and make a scatter plot of them using red dots.

Hold the graph, and determine those pairs that lie outside the triangle and make a scatter plot of them using blue dots, as shown above. Also, add the straight-line edges of the triangle to the graph.

Finally, determine the area of the triangle from the proportion of the red dots.

8. A skydiver jumps off a plane at a height of h_0 meters, with an initial vertical velocity v_0 (we will consider horizontal motion in another homework set.) The vertical drag force, discussed in Example 2.3 (p. 35) of the textbook, depends quadratically on the downward vertical velocity v :

$$F_{\text{drag}} = \frac{1}{2} \rho C A v^2$$



where ρ is the air density (assumed here to be independent of height), A is the effective area of the skydiver perpendicular to the motion, and C is a drag coefficient. Assuming a dense body and ignoring the buoyancy force,[†] the net downward vertical force on the skydiver is the difference between the force of gravity $F_{\text{grav}} = mg$, and the drag force, where m is the skydiver's mass plus equipment and g is the acceleration of gravity (also assumed to be independent of height). Thus, Newton's second law of motion ($F = ma$) states that:

$$m \frac{dv}{dt} = F_{\text{net}} = F_{\text{grav}} - F_{\text{drag}} = mg - \frac{1}{2} \rho C A v^2 \quad (1)$$

As the downward velocity v keeps increasing, the drag-force term in Eq. (1) keeps building up until it compensates the gravity force, resulting in zero acceleration, or, constant velocity v_c , referred to as *critical* or *terminal velocity*. From then on, the skydiver falls at that constant velocity. The balancing condition between gravity and drag force gives the value of the critical velocity:

$$mg - \frac{1}{2} \rho C A v_c^2 = 0 \quad \Rightarrow \quad v_c = \sqrt{\frac{2mg}{\rho C A}} \quad (2)$$

Let us define also the related quantities t_c and h_c ,

$$t_c = \frac{v_c}{g} = \sqrt{\frac{2m}{\rho C A g}}, \quad h_c = v_c t_c = \frac{v_c^2}{g} = \frac{2m}{\rho C A} \quad (3)$$

The skydiver can control the value of v_c by changing the effective area A . For example, if the skydiver turns vertical, then A decreases and v_c increases. Similarly, just before reaching ground, the skydiver opens a parachute, thus substantially increasing A and decreasing v_c . Using the definitions (2) and (3), Eq. (1) can be written in the simplified form:

$$\frac{dv}{dt} = \frac{v_c}{t_c} \left(1 - \frac{v^2}{v_c^2} \right) \quad (4)$$

The solution of the differential equation (4) with initial condition $v(t_0) = v_0$ is given by:

$$v(t) = v_c \frac{\frac{v_0}{v_c} + \tanh\left(\frac{t-t_0}{t_c}\right)}{1 + \frac{v_0}{v_c} \tanh\left(\frac{t-t_0}{t_c}\right)}, \quad t \geq t_0 \quad (5)$$

where t_c is a measure of the time constant to reach the critical velocity value.[‡] Note that $v(t_0) = v_0$ as it should, and $v(\infty) = v_c$. The solution (5) can be derived by standard calculus methods, or, by using

[†]From Archimedes' principle, the buoyancy force can be taken into account by replacing g by its effective value $g_{\text{eff}} = g(1 - \rho/\rho_{\text{obj}})$, where ρ_{obj} is the object's density. In this problem, we assume that $\rho \ll \rho_{\text{obj}}$.

[‡]Typically, $v(t)$ reaches about 99% of v_c within a couple of t_c 's while falling a distance of a couple of h_c 's.

MATLAB's symbolic math toolbox. You are not expected yet to know how to solve such differential equations, so in this homework, we'll just take the solution (5) as given. In a later homework, we will also solve it numerically using while-loops. The corresponding vertical drop distance y (measured from the airplane), can be obtained by integrating the above solution for v :

$$\frac{dy}{dt} = v \Rightarrow y(t) = h_c \ln \left[\cosh \left(\frac{t - t_0}{t_c} \right) + \frac{v_0}{v_c} \sinh \left(\frac{t - t_0}{t_c} \right) \right], \quad \text{for } t \geq t_0$$

Note that at $t = t_0$, we have $y(t_0) = 0$. The corresponding height measured from the ground (see above figure) is $h(t) = h_0 - y(t)$, or,

$$h(t) = h_0 - h_c \ln \left[\cosh \left(\frac{t - t_0}{t_c} \right) + \frac{v_0}{v_c} \sinh \left(\frac{t - t_0}{t_c} \right) \right], \quad t \geq t_0 \quad (6)$$

Often, we wish to know how long it takes to drop to a height $h \leq h_0$. This can be obtained by solving Eq. (6) for t in terms of h :

$$t = t_0 + \frac{h_0 - h}{v_c} + t_c \ln \left[\frac{v_c + \sqrt{v_c^2 + (v_0^2 - v_c^2) e^{-2(h_0 - h)/h_c}}}{v_c + v_0} \right], \quad h_0 \geq h \geq 0 \quad (7)$$

or, in terms of the drop distance $y = h_0 - h$,

$$t = t_0 + \frac{y}{v_c} + t_c \ln \left[\frac{v_c + \sqrt{v_c^2 + (v_0^2 - v_c^2) e^{-2y/h_c}}}{v_c + v_0} \right], \quad 0 \leq y \leq h_0 \quad (8)$$

As y increases by a few h_c lengths, or as h decreases towards zero, the term e^{-2y/h_c} becomes small and can be ignored, implying from Eq. (8) that the skydiver is then falling with constant terminal velocity v_c :

$$t \approx t_0 + \frac{y}{v_c} + t_c \ln \left[\frac{2v_c}{v_c + v_0} \right], \quad y \gg h_c$$

Setting $y = h_0$ gives the time it takes to reach the ground:

$$t_g = t_0 + \frac{h_0}{v_c} + t_c \ln \left[\frac{v_c + \sqrt{v_c^2 + (v_0^2 - v_c^2) e^{-2h_0/h_c}}}{v_c + v_0} \right]$$

Equations (5), (6), and (7) form the basis of this homework. Assume the following numerical values:

$\rho = 1.2$	kg/m ³ , air density
$g = 9.8$	m/sec ² , acceleration of gravity
$m = 70$	kg, skydiver's weight (mass)
$C = 1$	skydiver's drag coefficient

- a. Write a MATLAB function $V(t, t_0, v_0, v_c)$ that implements Eq. (5). It should be vectorized in the variable t , with t_0, v_0, v_c being parameters. Similarly, write functions $H(t, h_0, t_0, v_0, v_c)$ and $T(h, t_0, h_0, v_0, v_c)$ that implement Eqs. (6) and (7). The three functions must be defined as anonymous functions:

```
V = @(t,t0,v0,vc) ...
H = @(t,h0,t0,v0,vc) ...
T = @(h,t0,h0,v0,vc) ...
```

- b. Assume that the skydiver jumps from a height of $h_0 = 2500$ m, with zero initial velocity $v_0 = 0$, at $t_0 = 0$, and is oriented so that her effective surface area is $A_0 = 0.7$ m². Calculate the terminal

velocity v_{c0} , and then calculate the time t_1 it takes to drop to a height of $h_1 = 1500$ m above the ground and the speed v_1 at that time. Use the above functions for your calculations.

When the skydiver reaches the height h_1 , she suddenly changes orientation (e.g. turns sideways) so that her effective area is now $A_1 = 0.3$ m². Calculate the new terminal velocity v_{c1} . Use the values of v_1, t_1 as the initial conditions for the rest of the fall for $t \geq t_1$. Calculate the time t_2 at which the skydiver reaches a height of $h_2 = 200$ m above ground, and calculate the speed v_2 at that time instant.

At that time t_2 , the skydiver suddenly opens her parachute, which has surface area of $A_2 = 50$ m². Calculate the new terminal velocity v_{c2} .[†] Use the values of v_2, t_2 as the initial values for the rest of the fall for $t \geq t_2$. Calculate the time, say t_g , it takes to hit the ground (i.e., the height is $h = 0$.)

- c. Using the calculated values from part (b), and using appropriate relational operators and your function $V(t, t_0, v_0, v_c)$, define a single-line anonymous function $v(t)$ that describes the velocity of the fall through the various stages till the ground is hit, that is, define the function:

$$v(t) = \begin{cases} V(t, t_0, v_0, v_{c0}), & \text{if } t_0 \leq t \leq t_1 \\ V(t, t_1, v_1, v_{c1}), & \text{if } t_1 \leq t \leq t_2 \\ V(t, t_2, v_2, v_{c2}), & \text{if } t_2 \leq t \leq t_g \end{cases}$$

Similarly, define an overall height function:

$$h(t) = \begin{cases} H(t, h_0, t_0, v_0, v_{c0}), & \text{if } t_0 \leq t \leq t_1 \\ H(t, h_1, t_1, v_1, v_{c1}), & \text{if } t_1 \leq t \leq t_2 \\ H(t, h_2, t_2, v_2, v_{c2}), & \text{if } t_2 \leq t \leq t_g \end{cases}$$

Define the vector of time instants spanning the interval $0 \leq t \leq t_g$:

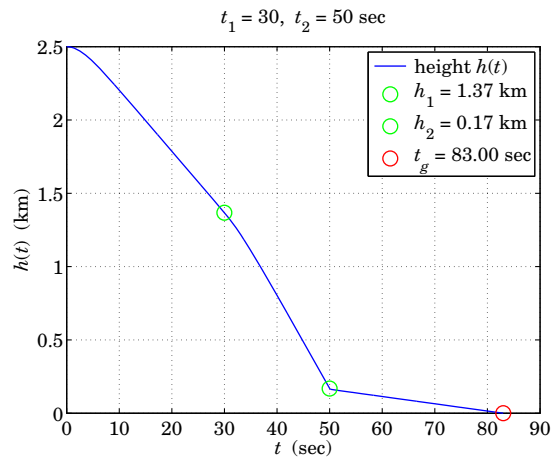
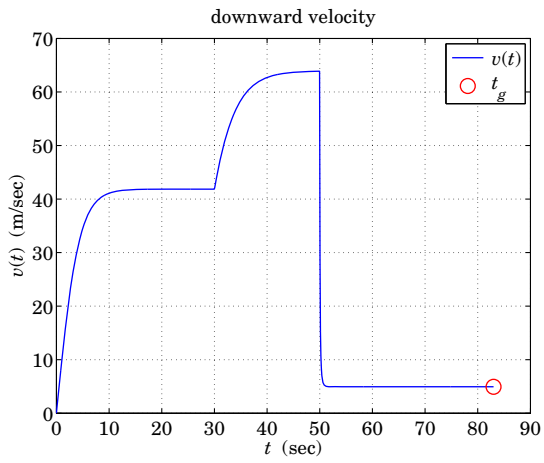
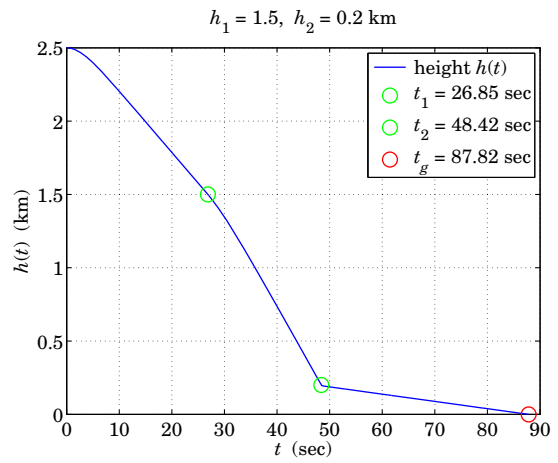
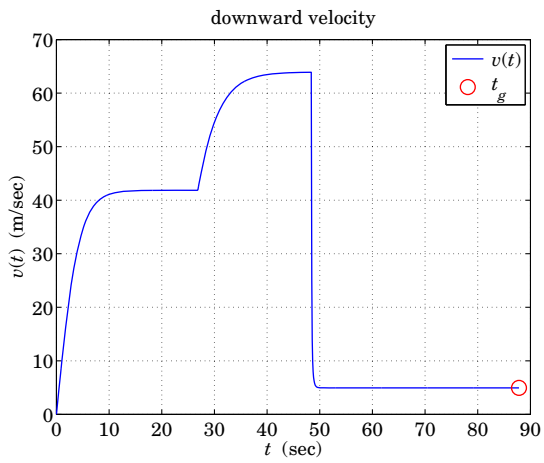
$$t = \text{linspace}(0, t_g, 1001);$$

Evaluate $v(t)$ and $h(t)$ at these times t , and plot them. Plot height in units of kilometers. See example plots below. On the height plot, place the points t_1, t_2, t_g at which the skydiver's configuration changes. Note that the height plots are almost, but not quite, straight-line plots with changing slopes because the terminal velocities change, see Eq. (7).

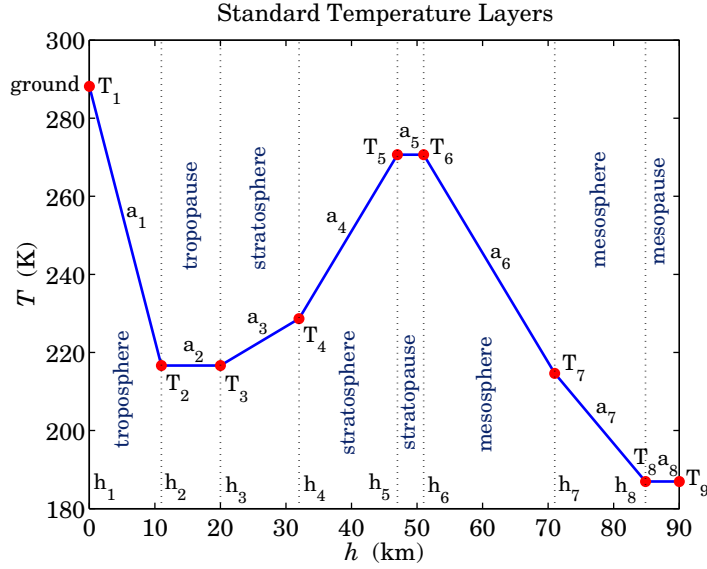
- d. In parts (b,c), the heights h_1, h_2 were given and you had to calculate the corresponding times t_1, t_2 at which changes in configuration took place.

In a slightly different version, assume now that these times are given to be $t_1 = 30$ and $t_2 = 50$ sec. Calculate the corresponding heights h_1, h_2 and the time t_g to reach the ground. Repeat the plots of part (c).

[†]The terminal velocity v_{c2} is roughly equal to the velocity of jumping off a height of about one meter.



9. *Standard Atmosphere.*[†] The standard atmosphere is a model of the average properties of the atmosphere that can be used to calculate representative temperature, air-pressure, and density distributions as functions of height up to about 90 km. The model assumes a certain piece-wise linear distribution of temperatures across the various layers of the atmosphere, such as the troposphere, stratosphere, and mesosphere and is shown in the figure below.



The standardized values of the heights h_i , temperatures T_i , and slopes a_i (referred to a lapse rates) of the various layers are given in the table below.

Layers	Heights (km)	Temperatures (K)	Slopes (K/km)
troposphere	$h_1 = 0$	$T_1 = 288.15$	$a_1 = -6.5$
tropopause	$h_2 = 11$	$T_2 = 216.65$	$a_2 = 0.0$
stratosphere	$h_3 = 20$	$T_3 = 216.65$	$a_3 = 1.0$
stratosphere	$h_4 = 32$	$T_4 = 228.65$	$a_4 = 2.8$
stratopause	$h_5 = 47$	$T_5 = 270.67$	$a_5 = 0.0$
mesosphere	$h_6 = 51$	$T_6 = 270.65$	$a_6 = -2.8$
mesosphere	$h_7 = 71$	$T_7 = 214.65$	$a_7 = -2.0$
mesopause	$h_8 = 84.852$	$T_8 = 186.946$	$a_8 = 0.0$
mesopause	$h_9 = 90$	$T_9 = 186.946$	

The heights h are actually, so-called *geopotential heights* that are related as follows to the actual geometrical heights z in terms of the Earth's radius, $R_e = 6356.766$ km, where h, z are in units of km:

$$h = \frac{R_e z}{R_e + z} \quad (9)$$

For example, the height $h_8 = 84.852$ km corresponds to a geometrical height of $z_8 = 86$ km. Because of the assumption of linearity, the given slopes are redundant and can be calculated from the relationships:

$$a_i = \frac{T_{i+1} - T_i}{h_{i+1} - h_i}, \quad i = 1, 2, \dots, 8$$

or, in a vectorized way using the **diff** function in MATLAB:

[†]<http://www.pdas.com/atmosdef.html>, see also, <http://www.pdas.com/atmos.html>

```

hi = [h1, h2, h3, h4, h5, h6, h7, h8, h9];
Ti = [T1, T2, T3, T4, T5, T6, T7, T8, T9];
ai = diff(Ti)./diff(hi);

```

Within the i -th layer with slope a_i , the temperature varies linearly as a function of height h :

$$h_i \leq h \leq h_{i+1} \Rightarrow T(h) = T_i + a_i(h - h_i), \quad (i\text{-th layer}) \quad (10)$$

Using the ideal gas law and Archimedes' principle of hydrostatic equilibrium, one obtains the following equations, which can be used to determine the pressure $P(h)$ and density $\rho(h)$ as functions of the geopotential height h :

$$P = \rho R_s T, \quad \frac{dP}{dh} = -g_0 \rho \quad (11)$$

where $R_s = 0.287053$ kJ/K/kg is the specific gas constant for air,[†] and $g_0 = 9.80665$ m/s² is the acceleration of gravity at the Earth's surface. Combining the two equations, we obtain,

$$\rho = \frac{P}{R_s T} \Rightarrow \frac{dP}{dh} = -\frac{g_0}{R_s} \frac{P}{T} \Rightarrow \frac{d}{dh} \ln(P(h)) = -\frac{B}{T(h)}$$

where we defined the constant B ,

$$B = \frac{g_0}{R_s} = 34.1632 \frac{\text{K}}{\text{km}} \quad (12)$$

Thus, within the i -th layer with slope a_i , we have

$$h_i \leq h \leq h_{i+1} \Rightarrow \frac{d}{dh} \ln(P(h)) = -\frac{B}{T_i + a_i(h - h_i)} \quad (13)$$

In integrating Eq. (13), we must distinguish the two cases $a_i \neq 0$ and $a_i = 0$. Denoting the pressure at $h = h_i$ by P_i , we obtain,

$$h_i \leq h \leq h_{i+1} \Rightarrow P(h) = \begin{cases} P_i \cdot \left(\frac{T_i}{T_i + a_i(h - h_i)} \right)^{B/a_i}, & \text{if } a_i \neq 0 \\ P_i \cdot \exp\left(-\frac{B(h - h_i)}{T_i}\right), & \text{if } a_i = 0 \end{cases} \quad (14)$$

In fact, the $a_i = 0$ case is the limit of the $a_i \neq 0$ case as $a_i \rightarrow 0$. This follows from the limiting form of the exponential function:

$$\lim_{\alpha \rightarrow 0} \left(\frac{1}{1 + \alpha x} \right)^{\frac{1}{\alpha}} = \exp(-x)$$

One might be tempted to define a MATLAB function that combines the two cases as

$$F = @(x, a) (1 + a*x).^(-1/a) * (a~=0) + \exp(-x) * (a==0);$$

that is,

$$f(x, \alpha) = \begin{cases} \left(\frac{1}{1 + \alpha x} \right)^{\frac{1}{\alpha}}, & \text{if } \alpha \neq 0 \\ \exp(-x), & \text{if } \alpha = 0 \end{cases}$$

However, the MATLAB function would produce a NaN when $\alpha = 0$ arising from the first term which would be $\infty \cdot 0$. The problem can be fixed by replacing $1/\alpha$ by its pseudo-inverse $\text{pinv}(\alpha)$,[‡] and define the function as follows,

[†] $R_s = R/M$, where $R = 8.31432$ J/K/mol is the ideal gas constant, and $M = 28.9644$ kg/kmol, the molecular weight of air.

[‡]the pseudoinverse of a scalar α is defined to be $1/\alpha$ if $\alpha \neq 0$, and 0, if $\alpha = 0$.

$$F = @(x,a) (1 + a*x).^(-pinv(a)) * (a~=0) + exp(-x) * (a==0);$$

With the help of the function $F(x, \alpha)$, we can express Eq. (14) in the compact form:

$$h_i \leq h \leq h_{i+1} \Rightarrow P(h) = P_i \cdot F\left(\frac{B(h - h_i)}{T_i}, \frac{a_i}{B}\right) \quad (15)$$

From Eq. (14), we can now obtain the pressure P_{i+1} at $h = h_{i+1}$, for $i = 1, 2, \dots, 8$:

$$P_{i+1} = P_i \cdot F\left(\frac{B(h_{i+1} - h_i)}{T_i}, \frac{a_i}{B}\right) = \begin{cases} P_i \cdot \left(\frac{T_i}{T_i + a_i(h_{i+1} - h_i)}\right)^{B/a_i}, & \text{if } a_i \neq 0 \\ P_i \cdot \exp\left(-\frac{B(h_{i+1} - h_i)}{T_i}\right), & \text{if } a_i = 0 \end{cases} \quad (16)$$

where the initial pressure at sea-level is taken to be $P_1 = 1 \text{ atm} = 101325 \text{ Pa}$. The corresponding densities at h_i are computed by

$$\rho_i = \frac{P_i}{R_s T_i}, \quad i = 1, 2, \dots, 9 \quad (17)$$

In particular, at sea level, $\rho_1 = P_1/R_s T_1 = 1.225 \text{ kg/m}^3$. Similarly, for h within the i -th layer, we find:

$$h_i \leq h \leq h_{i+1} \Rightarrow \rho(h) = \begin{cases} \rho_i \cdot \left(\frac{T_i}{T_i + a_i(h - h_i)}\right)^{1+B/a_i}, & \text{if } a_i \neq 0 \\ \rho_i \cdot \exp\left(-\frac{B(h - h_i)}{T_i}\right), & \text{if } a_i = 0 \end{cases} \quad (18)$$

- a. Using Eqs. (16) and (17), calculate the quantities $P_i, \rho_i, i = 1, 2, \dots, 9$. By inverting Eq. (9), calculate also the corresponding geometrical heights z_i in km, and using **fprintf** make a table exactly as shown below:

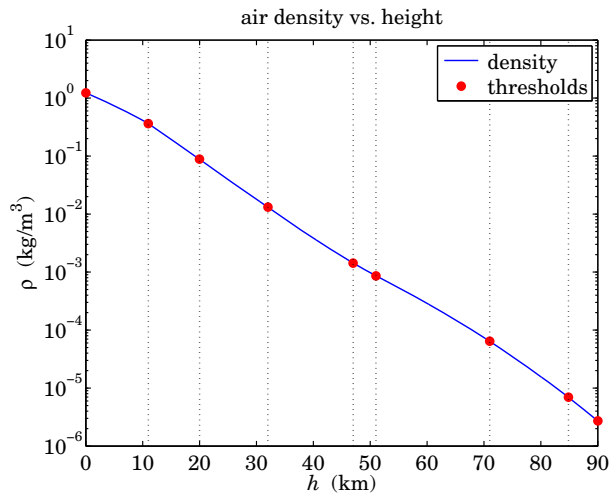
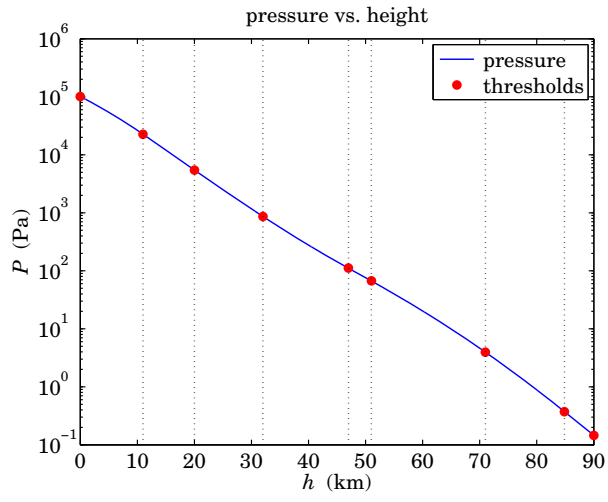
Atmospheric Layer	hi (km)	zi (km)	Ti (K)	ai (K/km)	Pi (Pa)	rhoi (kg/m ³)
troposphere	0.000	0.000	288.150	-6.5	101325.0000	1.225
tropopause	11.000	11.019	216.650	0.0	22632.0587	0.36392
stratosphere	20.000	20.063	216.650	1.0	5474.8862	0.088035
stratosphere	32.000	32.162	228.650	2.8	868.0180	0.013225
stratopause	47.000	47.350	270.650	0.0	110.9062	0.0014275
mesosphere	51.000	51.412	270.650	-2.8	66.9388	0.0008616
mesosphere	71.000	71.802	214.650	-2.0	3.9564	6.4211e-005
mesopause	84.852	86.000	186.946	0.0	0.3734	6.9579e-006
mesopause	90.000	91.293	186.946	0.0	0.1457	2.7159e-006

- b. Using the above values of P_i, ρ_i, T_i, a_i , and appropriate relational operators, write three single-line anonymous functions to calculate the temperature $T(h)$, pressure $P(h)$, and density $\rho(h)$ at any vector of heights in the range $0 \leq h \leq 90 \text{ km}$. For example, $T(h)$ is a piece-wise function with eight pieces, whose i -th piece is defined by Eq. (10). We may express it symbolically with the help of MATLAB's relational operators:

$$T(h) = \sum_{i=1}^8 [T_i + a_i(h - h_i)] \cdot (h \geq h_i \ \& \ h \leq h_{i+1})$$

We may similarly construct $P(h)$ and $\rho(h)$ whose i -th pieces are defined in Eqs. (14) and (18), respectively. You may also define $\rho(h) = P(h)/(T(h) \cdot R_s)$, in terms of $P(h)$ and $T(h)$.

- c. Define a vector $h = 0 : 0.1 : 90$ and evaluate and plot the functions $T(h), P(h)$, and $\rho(h)$. Make a plot of $T(h)$ like the one shown at the beginning. Make a **semilogy** plot for $P(h)$ and $\rho(h)$ as shown below.



Homework Problems – Week 8
440:127 – Spring 2015 – S. J. Orfanidis

1. End-of-Chapter Problem 9.10.
2. End-of-Chapter Problem 9.11. To clarify, the problem is asking you to determine the cost of the last four years out of the 22 years of planning.
3. End-of-Chapter Problem 9.13. You must use a while-loop to implement your function.
4. Consider the MATLAB relational operations:

```
m = find(a)
k = (a ~= 0)
```

where a is a row vector. The objective of this problem is to reverse-engineer these operations. Write a MATLAB function,

```
[m,k] = my_find(a);
```

that implements the above relational operations using for-loops and if-statements *only* and does not use the function **find**. The outputs m, k must be the same as those above, and k must be of type **logical**. Test it on the case:

```
a = [2 0 4 0 0 5 0 7];
```

Verify that $a(k)$ and $a(m)$ return the same output. Write two other versions of this function that use: (i) conventional while-loops, and (ii) forever while-loops, and test them on the same input.

5. Write your own version of the **max** function that uses for-loops to determine the maximum of an array, as well as the index at which that maximum occurs. It must have usage:
with usage:

```
[ymax, imax] = my_max(y)
```

Test it on the following vector $y = [1, -2, 3, 8, -7, 2]$. How would you use your function to determine the minimum of the array y , and the index at which it occurs?

6. Consider the finite sum:

$$S = \sum_{k=0}^N (3k + 2)^2 = (3 \cdot 0 + 2)^2 + (3 \cdot 1 + 2)^2 + (3 \cdot 2 + 2)^2 + \cdots + (3 \cdot N + 2)^2 \quad (1)$$

- a. Use a for-loop to evaluate the above sum for $N = 100$.
- b. Use a conventional while-loop to evaluate the above sum for $N = 100$.
- c. Use a forever while-loop to evaluate the above sum for $N = 100$.
- d. Write a function, $S = f(N)$, as an M-file, that calculates the above sum for any integer value $N \geq 0$ using a for-loop. It can be verified that the above sum is given analytically by the following expression:

$$S = g(N) = (N + 1)(3N^2 + 7.5N + 4) \quad (2)$$

Write an anonymous function that implements Eq. (2). Then verify that your $f(N)$ and $g(N)$ produce the same answers for $N = 100$.

- e. Using a forever while-loop, determine the largest N such that $f(N) \leq S_{\max}$, where $S_{\max} = 10^7$.

- f. Determine the required N of part (e) by solving the equation $f(N) = S_{\max}$ using the function **fzero**. (You need to round down the resulting solution from **fzero**).
- g. Multiply out the factors in Eq. (2) to express $g(N)$ as a cubic polynomial in N . Look up the built-in function **roots** and use it to determine the required value of N of part (e) by solving the cubic equation $g(N) = S_{\max}$. [Hint: Pick the real-valued root among the three roots and round it down.]
7. *Thermostat Model.*[†] A typical home furnace supplies an amount of heat that increases the air temperature of a room by $R_0 = 20$ °F per hour. The time rate of change of the room temperature is governed by Newton's law of cooling:

$$\frac{dT(t)}{dt} = -k[T(t) - T_{\text{ext}}(t)] + R(t) \quad (3)$$

where $T(t)$ is the room temperature at time t , $T_{\text{ext}}(t)$ is the external temperature, k is a measure of the loss of heat through the walls, and $R(t)$ is the rate of temperature increase per hour supplied by the furnace (like the R_0 above.) A typical home thermostat can be programmed to several temperature settings during the day. Here, we will assume two settings, a higher temperature setting T_H for the first 12 hours of a day, and a lower setting T_L for the second 12 hours. Thus, the control temperature of the thermostat is defined by the time function:

$$T_c(t) = \begin{cases} T_H, & \text{if } \text{mod}(t, 24) < 12 \\ T_L, & \text{if } \text{mod}(t, 24) \geq 12 \end{cases} \quad (4)$$

where the modulo operation, $\text{mod}(t, 24)$, reduces the time t modulo 24, i.e., it finds the remainder of the division of t by 24, so that it is always in the range $0 \leq \text{mod}(t, 24) < 24$. For example, for a 48-hour period, we will have:

t =	0	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
mod(t, 24) =	0	1	2	3	4	5	6	7	8	9	10	11
	12	13	14	15	16	17	18	19	20	21	22	23
	0	1	2	3	4	5	6	7	8	9	10	11
	12	13	14	15	16	17	18	19	20	21	22	23

If the room temperature falls below the prescribed control temperature (T_H or T_L), the thermostat turns the furnace on until the control temperature is reached and then it turns the furnace off. This can be modeled into Eq. (3) by choosing the control signal $R(t)$ as follows:

$$R(t) = \begin{cases} R_0, & \text{if } T(t) < T_c(t) \\ 0, & \text{if } T(t) \geq T_c(t) \end{cases} \quad (5)$$

Because $R(t)$ depends on $T(t)$ in a nonlinear manner, Eq. (3) can only be solved numerically, and this is the main objective of this problem. To this end, time is discretized in small equal-step increments, $t_n = n\Delta t$, $n = 1, 2, 3, \dots$, where Δt is a small step size. The time-derivative in Eq. (3) can be approximated as a ratio of differences, resulting in the following difference equation:

$$\frac{T(t_{n+1}) - T(t_n)}{\Delta t} = -k[T(t_n) - T_{\text{ext}}(t_n)] + R(t_n)$$

[†]For a more realistic version see the paper by P. S. Sansgiry and C. C. Edwards, "A Home Heating Model for Calculus Students," *Coll. Math. J.*, 27, 395 (1996), placed on sakai.

Using the simplified notation $T(n)$ to denote $T(t_n)$, and similarly for $R(t_n)$ and $T_c(t_n)$, this difference equation can be rearranged into:

$$T(n+1) = T(n) - k \Delta t [T(n) - T_{\text{ext}}(n)] + \Delta t R(n), \quad n \geq 1 \quad (6)$$

where

$$R(n) = \begin{cases} R_0, & \text{if } T(n) < T_c(n) \\ 0, & \text{if } T(n) \geq T_c(n) \end{cases} \quad \text{with} \quad T_c(n) = \begin{cases} T_H, & \text{if } \text{mod}(t_n, 24) < 12 \\ T_L, & \text{if } \text{mod}(t_n, 24) \geq 12 \end{cases} \quad (7)$$

The initial value in Eq. (6) will be assumed given, i.e., $T(1) = T_0$. For the external temperature, we will assume a simple sinusoidal model with 24-hr periodicity:

$$T_{\text{ext}}(n) = A - B \cos\left(\frac{2\pi t_n}{24}\right) \quad (8)$$

Consider the following realistic numerical values:

$$A = 40 \text{ }^\circ\text{F}, \quad B = 10 \text{ }^\circ\text{F}$$

$$k = 0.35 \text{ hr}^{-1}, \quad R_0 = 20 \text{ }^\circ\text{F/hr}$$

$$T_H = 70 \text{ }^\circ\text{F}, \quad T_L = 60 \text{ }^\circ\text{F}, \quad T_0 = 35 \text{ }^\circ\text{F}$$

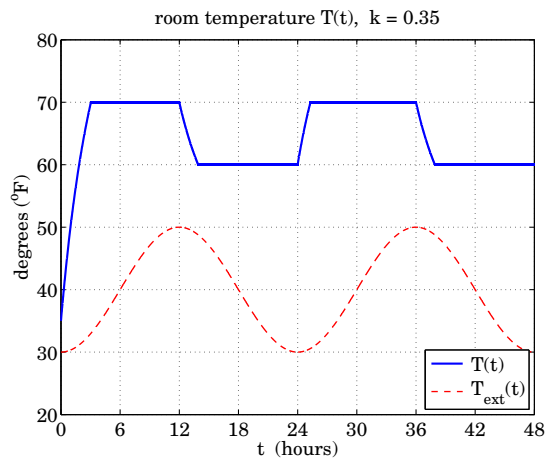
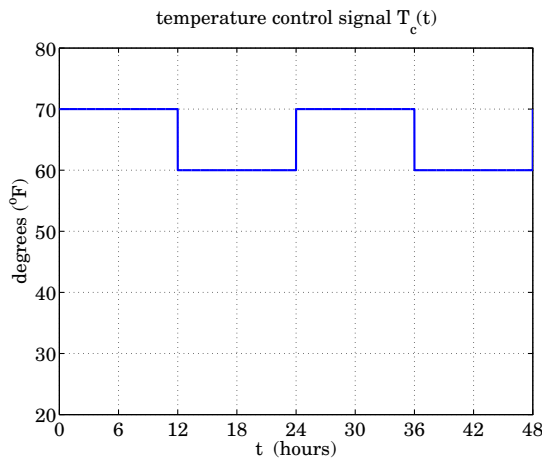
Define the time vector t_n to span a 48-hr period and sampled every 3 seconds:

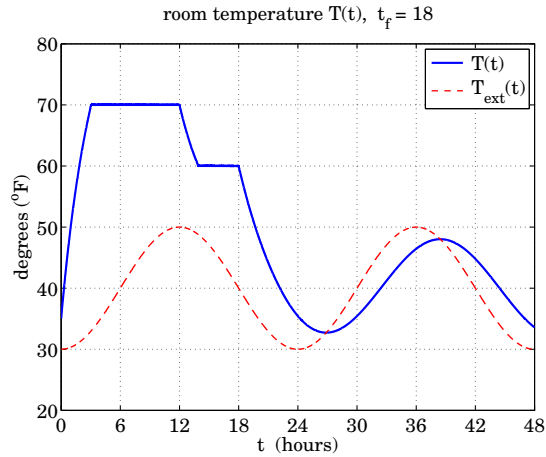
```
Tmax = 48; Dt = 3/3600;      % units of hours
tn = Dt:Dt:Tmax;
```

- Use a for-loop to calculate the control signal $T_c(n)$ and plot it versus t_n . Within the same for-loop, calculate also the actual room temperature $T(n)$, and on separate graph plot it versus t_n together with the external temperature $T_{\text{ext}}(n)$.

Observe the initial transients starting from T_0 , and the ability of the thermostat system to follow the prescribed high/low settings, switching between the two at every 12-hr period.

- Repeat the calculation and plotting of $T(n)$ using the value $k = 0.25$, corresponding to a well-insulated house, and then using $k = 0.50$ for a poorly insulated one.
- For the case $k = 0.35$, assume that there is a power failure at time $t_f = 18$ and that from then on the furnace stops operating. Calculate and plot the room temperature and observe how it eventually follows the external temperature variations (with some lag.)





8. *Skydiver*.[†] Continuing with last week’s Problem 8, we consider now the issue of horizontal motion that we ignored then. At the instant of jumping, the horizontal velocity of the skydiver is equal to that of the airplane. We will see below that the air drag in the horizontal direction quickly slows down the horizontal motion to zero and the skydiver effectively continues to fall vertically as we assumed in last week’s problem. We recall that the force due to the air resistance acting against a moving object is taken to be proportional to the square of the object’s velocity, typically at speeds less than 200 mph:

$$F = \frac{1}{2} \rho C A v^2$$

where C, ρ, A are the drag coefficient, air density, and object’s cross-sectional area. The direction of the force is opposite that of the velocity. To simplify the subsequent notation, let us define the following “drag constant” D , where m is the skydiver’s mass:

$$D = \frac{\rho C A}{2m}$$

Newton’s equations of motion that take into account both horizontal and vertical motions are as follows, where x, y denote the horizontal and vertical distances (y is measured downwards from the airplane), and v_x, v_y are the corresponding velocities, and a_x, a_y , the accelerations:

$$\begin{aligned}
 v &= \sqrt{v_x^2 + v_y^2} && \text{(velocity magnitude)} \\
 v_x &= \frac{dx}{dt}, \quad a_x = \frac{dv_x}{dt} = -D v_x v && \text{(horizontal drag)} \\
 v_y &= \frac{dy}{dt}, \quad a_y = \frac{dv_y}{dt} = -D v_y v + g && \text{(vertical drag opposing gravity)}
 \end{aligned} \tag{9}$$

where g is the acceleration of gravity acting vertically downwards. These differential equations may be solved numerically using, for example, MATLAB’s built-in differential equation solver **ode45**. However, in this problem, we are going to replace them with a discrete-time version that can accurately determine the solution in an iterative manner using a while-loop. The homework solutions will demonstrate the **ode45** solution, which turns out to be virtually indistinguishable from the one presented here.

We assume that time is discretized in small steps $t_n = (n - 1)T$, $n = 1, 2, 3, \dots$, where T is a very small step increment. Let us denote by $x(n)$ the value of the horizontal distance $x(t_n)$ at time $t = t_n$, and similarly for the quantities $y(n), v_x(n), v_y(n)$.[‡] Then, Eq. (9) can be replaced by the following

[†]G. Wagner and R. Wood, “Skydiver Survives Death Plunge (and the physics that helped)”, *Phys. Teacher*, **34**, 543 (1996), on sakai.
[‡] n can be thought of as a MATLAB index and $x(n)$ as a MATLAB array.

discretized version, where the operations must be done in the indicated order:*

$$\begin{aligned}
v(n) &= \sqrt{v_x^2(n) + v_y^2(n)} \\
a_x(n) &= -D v_x(n) v(n) \\
a_y(n) &= -D v_y(n) v(n) + g \\
x(n+1) &= x(n) + T v_x(n) \\
v_x(n+1) &= v_x(n) + T a_x(n) \\
y(n+1) &= y(n) + T v_y(n) \\
v_y(n+1) &= v_y(n) + T a_y(n)
\end{aligned} \tag{10}$$

This works as follows: At time n , we assume that we know the quantities $x(n), y(n), v_x(n), v_y(n)$. From Eq. (10), we first calculate the accelerations $a_x(n), a_y(n)$, and then use them to calculate the next values $x(n+1), y(n+1), v_x(n+1), v_y(n+1)$, and the process is repeated. To get the recursions started we may take the initial values to be at $n = 1$

$$x(1) = 0, \quad v_x(1) = v_0, \quad y(1) = 0, \quad v_y(1) = 0 \tag{11}$$

where v_0 is the initial horizontal velocity. The initial vertical velocity is assumed to be zero. If h_0 is the initial height of the jump, then the recursions (10) are to be iterated until the skydiver reaches the ground, i.e., until $y(n) \approx h_0$. The last index n determines the total time to reach ground.

To make the problem more realistic, we will assume that when the skydiver pulls the rip cord, the parachute does not open instantaneously (as in last week's problem), but rather it takes a few seconds to fully open. We can model the opening of the parachute by the following gradual increase of its surface area over time lasting from t_1 to t_2 :[†]

$$A(t) = \frac{1}{2}(A_2 - A_1) \cdot \tanh\left(10 \frac{t - t_{\text{mid}}}{t_2 - t_1}\right) + \frac{1}{2}(A_2 + A_1) \tag{12}$$

where $t_{\text{mid}} = (t_1 + t_2)/2$. Effectively, for $t \leq t_1$ the surface area is A_1 , and for $t \geq t_2$, it is A_2 , and during $t_1 < t < t_2$, it gradually transitions from A_1 to A_2 , see the last graph below. By contrast, a sudden change in the area can be modeled relative to the mid time-point t_{mid} as follows:

$$A(t) = \begin{cases} A_1, & \text{if } t \leq t_{\text{mid}} \\ A_2, & \text{if } t > t_{\text{mid}} \end{cases} \tag{13}$$

The time-varying area $A(t)$ defines effectively a time-varying drag constant $D(t)$ whose sampled values $D(t_n)$ at time $t = t_n$ must be used in the difference equations (10):

$$D(t) = \frac{\rho C A(t)}{2m} \tag{14}$$

In the sequel, we assume the following values of the parameters:

*We are assuming that the drag coefficient C is the same in the vertical and horizontal directions (an oversimplifying assumption).

[†]Note, $\tanh(\pm 5) = \pm 0.9999$.

$\rho = 1.2 \text{ kg/m}^3$	air density
$g = 9.81 \text{ m/sec}^2$	acceleration of gravity
$m = 77 \text{ kg}$	skydiver's mass plus equipment
$C = 1$	drag coefficient, typically $C = 0.2$ - 1.4 , depending on orientation
$A_1 = 0.7 \text{ m}^2$	area with parachute closed
$A_2 = 50 \text{ m}^2$	area with parachute opened
$t_1 = 50 \text{ sec}$	time when rip cord is pulled
$t_2 = 70 \text{ sec}$	time when parachute is fully opened
$v_0 = 50 \text{ m/sec}$	initial horizontal velocity, 112 mph
$h_0 = 2.5 \text{ km}$	initial height
$T = 0.005 \text{ sec}$	time-step increment

- For the given values above, define a single-line anonymous function that implements the area $A(t)$ as a function of time t using the definition (12). Define also a function for the drag constant $D(t)$. Plot $A(t)$ over the interval $0 \leq t \leq 120 \text{ sec}$. Indicate on the graph the points at $t = t_1$ and $t = t_2$.
- Initialize the arrays $x(n), y(n), v_x(n), v_y(n)$ as in Eq. (11), and for the above value of T , run a *forever* while-loop that calculates the arrays $a_x(n), a_y(n), v_x(n), v_y(n), x(n), y(n)$, and insert a condition that breaks out of the loop when $y(n)$ becomes just greater than h_0 , i.e., when the skydiver has reached the ground. Use the time-varying drag constant $D(t_n)$ at each time instant. Your loop should have following structure:

```

n=1; % initialize time index

while 1 % forever while-loop
    if y(n) > h0, break; end % break when ground is reached
    tn = (n-1)*T; % n-th time instant in seconds
    t(n) = tn; % build time vector - needed for plots
    Dn = D(tn); % drag constant at time tn
    v = sqrt(vx(n)^2 + vy(n)^2); % no need to save v in an array
    ax(n) = -Dn * vx(n) * v; % horizontal acceleration
    ... etc ...
    n = n+1; % update time index
end

```

After exiting the loop, delete the last values of the arrays x, y, v_x, v_y , but not those of t, a_x, a_y . Explain why this is necessary. What are the lengths of the resulting arrays?

- From the last value of t , determine the total duration t_g in seconds of the skydiver's fall until she reaches the ground.

Define the corresponding array of heights measured from ground, that is, $h(n) = h_0 - y(n)$, and calculate the value of the height h_g at time $t = t_g$. This height should be extremely small but it's not quite zero because the loop was exited prematurely—one more iteration would have made h_g slightly negative. In other words, for the last n , we have $h(n) \geq 0$, but $h(n+1) < 0$, or equivalently, $y(n) \leq h_0$, but $y(n+1) > h_0$.

Calculate also the total horizontal distance x_g at ground level. Moreover calculate the array indices n_1 and n_2 corresponding to the times t_1 and t_2 — for example, use the formula $t_1 = (n_1 - 1)T$ and solve it for n_1 , rounding the answer to the nearest integer.

- d. Calculate the vertical terminal velocities when the parachute is closed and when it is opened, that is, corresponding to the two surface areas A_1, A_2 . Use the formula from last week's Problem 8:

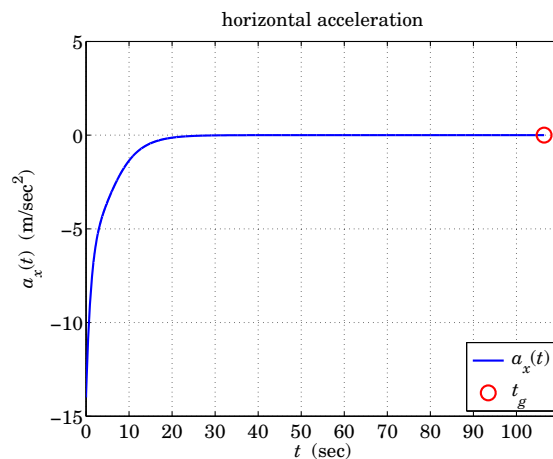
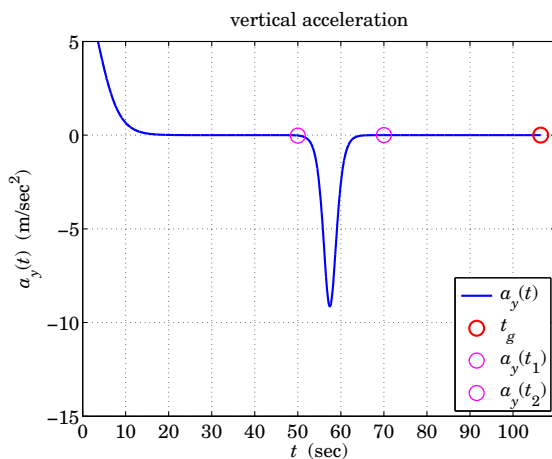
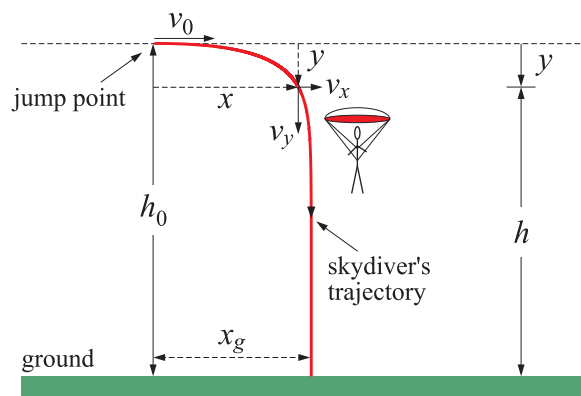
$$v_c = \sqrt{\frac{g}{D}} = \sqrt{\frac{2mg}{\rho CA}}$$

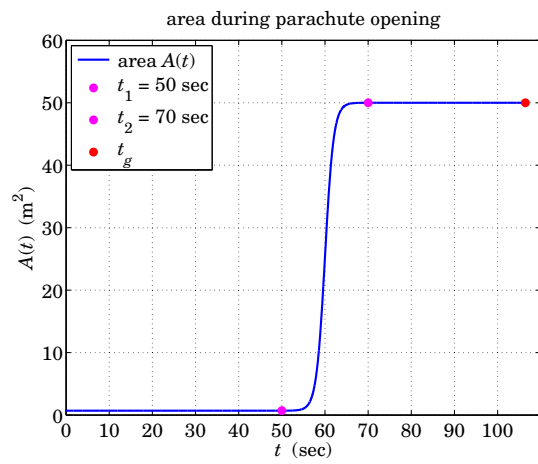
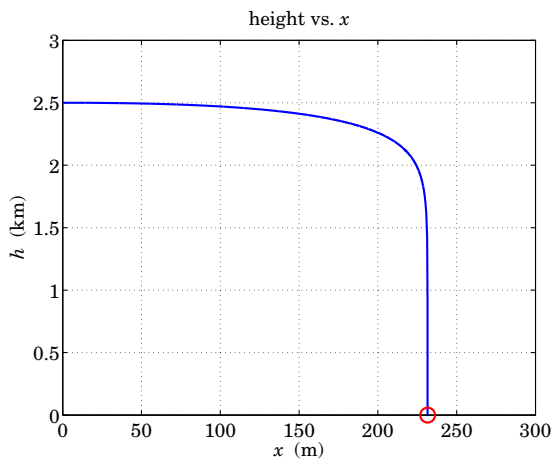
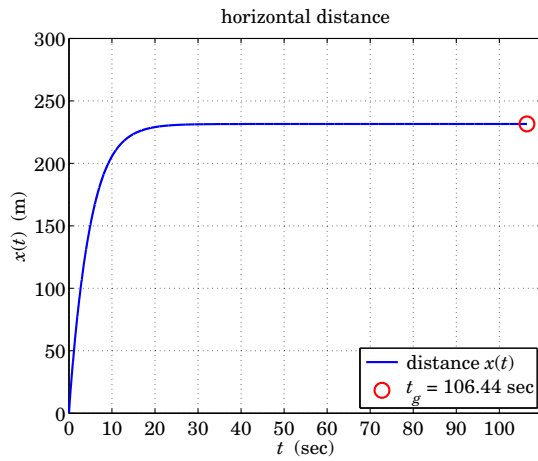
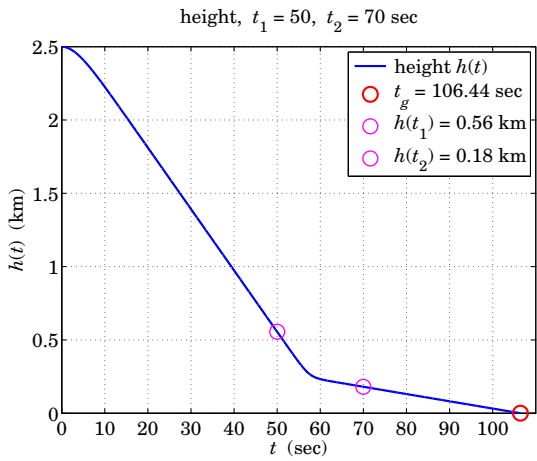
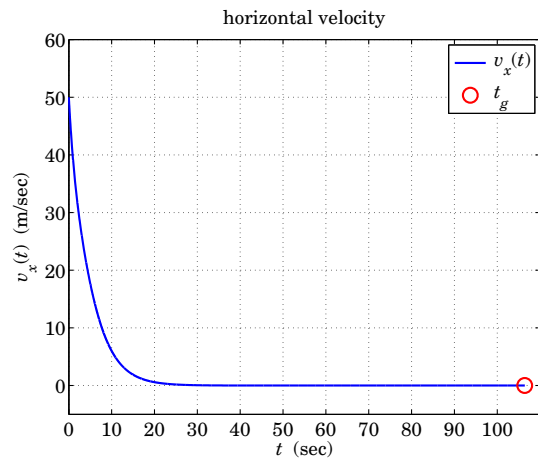
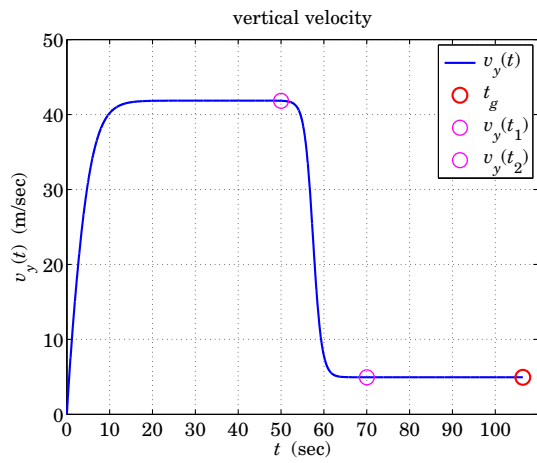
and compare them with the values of the vertical velocities at times $t = 30$ and $t = 80$ sec (see the graph below for the justification of these choices.)

- e. Make the following plots, and on all the graphs place the points corresponding to the time instants t_1 and t_2 , indicating the pulling of the rip cord and when the parachute is completely opened:

- vertical and horizontal accelerations $a_y(t), a_x(t)$ versus t , for $0 \leq t \leq t_g$.
- vertical and horizontal velocities $v_y(t), v_x(t)$ versus t , for $0 \leq t \leq t_g$.
- height $h(t)$ and horizontal distance $x(t)$ versus t , for $0 \leq t \leq t_g$.
- height h versus horizontal distance x , for $0 \leq x \leq x_g$, (it quickly becomes a vertical fall.)

- e. Repeat parts (a-e), by using the area function of Eq. (13) instead of Eq. (12).





9. *Epidemiological Modeling of Social Networks*. A recent paper,[†] predicting the demise of Facebook in a couple of years, has generated a lot of attention. It applies a modified version of the SIR model of infectious diseases to social networks. The model is verified first on the already demised Myspace social network, and then it is applied on the current status of Facebook. In this problem, we reproduce the results of this paper by using the optimum estimated model parameters and iterating a discrete-time version of the model's differential equations using a for-loop.

The epidemiological model for a social network is described by the following system of coupled differential equations for the quantities $S(t), I(t), R(t)$, where b, c are constant parameters:

$$\begin{array}{l} \frac{dS}{dt} = -bIS \\ \frac{dI}{dt} = bIS - cIR \\ \frac{dR}{dt} = cIR \end{array} \quad \begin{array}{l} S(t) = \text{no. of people "susceptible" to joining the network} \\ I(t) = \text{no. of "infected" people that have joined the network} \\ R(t) = \text{no. of "recovered" people that have left the network} \end{array} \quad (15)$$

Assuming equally-spaced time instants, $t_n = (n-1)T$, where T is a small time-step, we may approximate Eqs. (15) by the following system of discrete-time coupled difference equations, where $S(n)$ denotes the value $S(t_n)$:

$$\begin{array}{l} S(n+1) = S(n) - T b I(n) S(n) \\ I(n+1) = I(n) + T b I(n) S(n) - T c I(n) R(n) \\ R(n+1) = R(n) + T c I(n) R(n) \end{array} \quad (16)$$

These are obtained by approximating the derivatives in Eq. (15) as follows:

$$\frac{dS(t_n)}{dt} \approx \frac{S(n+1) - S(n)}{T}$$

We note that the sum $S(t) + I(t) + R(t)$, or $S(n) + I(n) + R(n)$ in the discrete case, remains independent of time. Let N denote this sum. Its value will depend on the assumed initial conditions for S, I, R .

- a. Let us test the model of Eq. (16) on the Myspace case first. The data file **myspace.dat** contains Google weekly query data for the keyword "myspace". Such query data serve as proxies for the number $I(t)$ of people that have joined the network at time t . The data span the period from Jan. 2004 to Feb. 2014. The second column of the data file contains 525 weekly query data points normalized so that the highest number is 100. The first column is the time in units of years, which can be parametrized as follows, assuming 52 weeks per year:

$$t_{\text{year}} = 2004 + \frac{t}{52}, \quad t = 0, 1, 2, \dots, 524, \quad (t = \text{time in weeks}) \quad (17)$$

The optimum values of the parameters b, c and the initial values S_0, I_0, R_0 were estimated in the above paper by using a least-squares fitting method. The obtained values were:

$$\begin{array}{l} N = 92.94 \\ R_0 = N \cdot 7.19 \cdot 10^{-3} \\ I_0 = N \cdot 9.49 \cdot 10^{-4} \\ S_0 = N - I_0 - R_0 \end{array} \quad \text{and} \quad \begin{array}{l} \beta = 5.98 \cdot 10^{-2}, \quad b = \frac{\beta}{N} \\ \nu = 2.68 \cdot 10^{-2}, \quad c = \frac{\nu}{N} \end{array}$$

[†]J. Cannarella and J. A. Spechler, "Epidemiological modeling of online social network dynamics", *arXiv*: 1401.4208, <http://arxiv.org/abs/1401.4208v1>, placed on sakai

The values of the parameters b, c assume that time was in units of weeks. However, in plotting the results, the time scale will be in years as in Eq. (17). Starting with the initial values,

$$\begin{bmatrix} S(1) \\ I(1) \\ R(1) \end{bmatrix} = \begin{bmatrix} S_0 \\ I_0 \\ R_0 \end{bmatrix}$$

iterate Eq. (16) for $n = 1 : 521$, spanning in months the 10-year period from Jan. 2004 to Jan. 2014. Since the time scale is months, we may choose the time step to be one month, i.e., $T = 1$. However, we will use the value $T = 1.02$ which works much better in matching the solutions of the two versions of Eqs. (15) and (16).[†]

Plot the iterative solution for $I(n)$ versus time in years, and add the observed data on the graph. The model describes the rise and fall of Myspace very well.

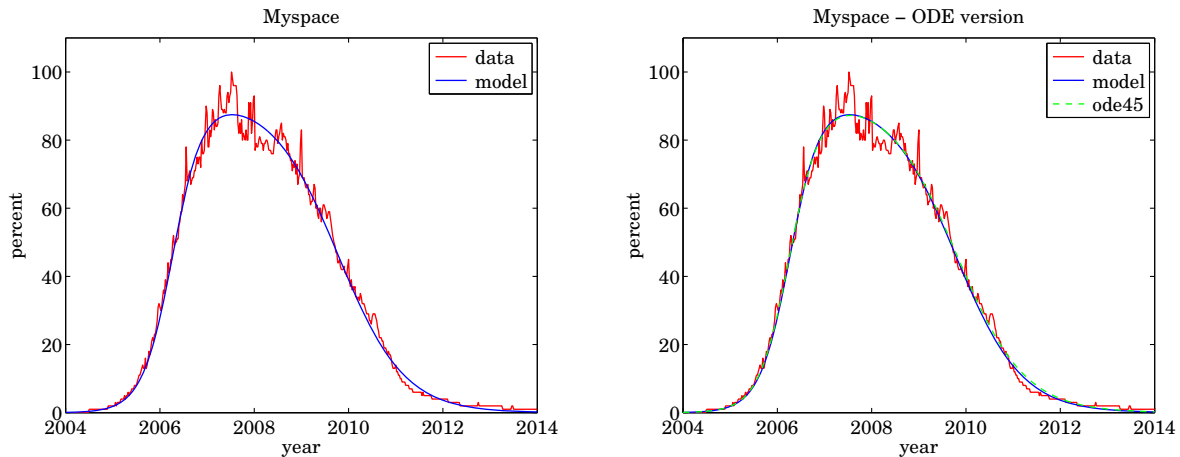
- b. This an optional part. Based on the summary of Matlab differential equation solvers given in the Appendix, the following Matlab function can be used to define the differential equation system (15),

$$f = @(t,z) [-b*z(2)*z(1); b*z(2)*z(1) - c*z(2)*z(3); c*z(2)*z(3)];$$

Using this function and defining the time-span to be the same as in part (a), that is,

$$tspan = 0:521;$$

integrate Eq. (15) and plot $I(t)$ vs. t (in years), and add the solutions of part (a) on the same graph, as shown below.



- c. Next, load the Facebook data from the file **facebook.dat**, which span the period from Jan. 2006 to Feb. 2014. Again, the first column is the time in years, and the second are the observed values of the variable I . The optimum model parameters determined in the above paper are in this case:

$$\begin{aligned} N &= 94.5 \\ R_0 &= N \cdot 2.35 \cdot 10^{-6} \\ I_0 &= N \cdot 6.43 \cdot 10^{-5} \\ S_0 &= N - I_0 - R_0 \end{aligned} \quad \text{and} \quad \begin{aligned} \beta &= 3.36 \cdot 10^{-2}, \quad b = \frac{\beta}{N} \\ \nu &= 4.98 \cdot 10^{-2}, \quad c = \frac{\nu}{N} \end{aligned}$$

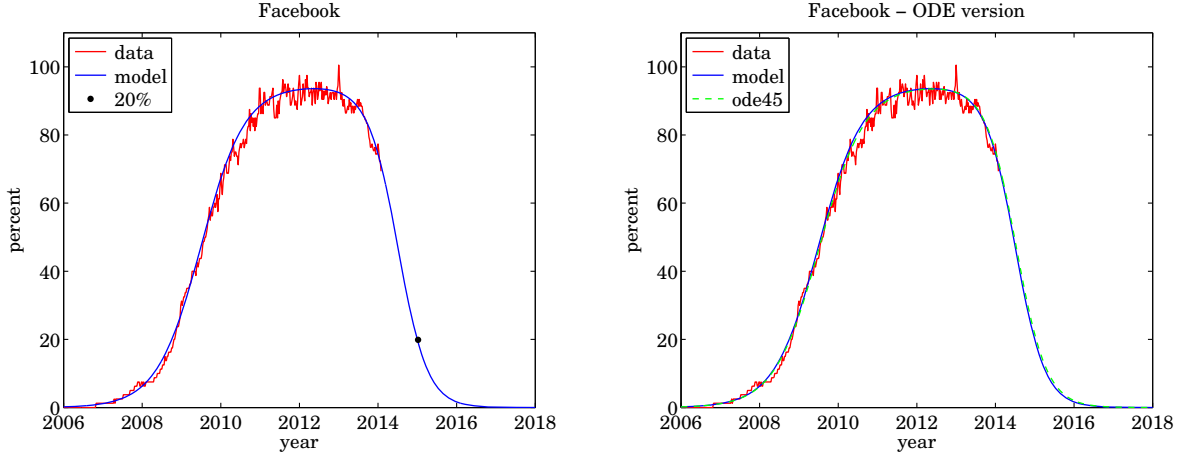
Repeat parts (a,b). However, now we wish to extrapolate the model into the future and predict the status of Facebook to Jan. 2018. Therefore, we will iterate Eq. (16) over the 14-year span from Jan. 2004 to Jan. 2018, corresponding to the time index $n = 1 : 729$ months. The time vector in years is still defined by Eq. (17) with t going up to $t = 728$. Again, use $T = 0.02$ for the time-step.

Plot the observed data, together with the calculated model prediction $I(n)$ from Eq. (16), but adjust the time scale to display only the years from Jan. 2006 to Jan. 2018.

[†] this is equivalent to choosing $T = 1$ but changing slightly the values of b, c in (16)

Note that the model matches the observed data very well until the present time of Feb. 2014, and predicts a rapid downfall in the number of Facebook users beyond that.[†]

Using appropriate relational operations, determine the time (month and year) at which the value of $I(n)$ will have fallen to 20 percent, and place it on the graph.



Appendix

Quick Introduction to MATLAB's Differential Equation Solvers

MATLAB has several ordinary differential equation (ODE) solvers, such as `ode45` or `ode23`, descriptions of which can be found by the command `'doc ode23'`. All higher-order ODEs can be reformulated as a system of first-order ODEs of the form:

$$\dot{z} = f(t, z) \quad (18)$$

where z is a column vector of time functions $z(t)$ and \dot{z} denotes the time-derivative dz/dt . For example, the system of Eqs. (9) is already first-order in the variables $\{x, v_x, y, v_y\}$. It can be put into the form of Eq. (18) by defining the vector z by

$$z = \begin{bmatrix} x \\ v_x \\ y \\ v_y \end{bmatrix} \Rightarrow \dot{z} = \begin{bmatrix} \dot{x} \\ \dot{v}_x \\ \dot{y} \\ \dot{v}_y \end{bmatrix} = \begin{bmatrix} v_x \\ -D v_x v \\ v_y \\ g - D v_y v \end{bmatrix} = \begin{bmatrix} v_x \\ -D v_x \sqrt{v_x^2 + v_y^2} \\ v_y \\ g - D v_y \sqrt{v_x^2 + v_y^2} \end{bmatrix} \quad (19)$$

Because $z(1), z(2), z(3), z(4)$ are identified as x, v_x, y, v_y , we can rewrite (19) in a form that defines the vector-valued function $f(t, z)$, allowing also for the time-variation of D :

$$\begin{bmatrix} \dot{z}(1) \\ \dot{z}(2) \\ \dot{z}(3) \\ \dot{z}(4) \end{bmatrix} = \dot{z} = f(t, z) = \begin{bmatrix} z(2) \\ -D(t) z(2) \sqrt{z(2)^2 + z(4)^2} \\ z(4) \\ g - D(t) z(4) \sqrt{z(2)^2 + z(4)^2} \end{bmatrix} \quad (20)$$

Assuming that the parameter g and the function $D(t)$ have been previously defined, then one can define the $f(t, z)$ function in MATLAB as an anonymous function:

```
f = @(t,z) [z(2); -D(t)*z(2)*sqrt(z(2)^2 + z(4)^2); z(4); g-D(t)*z(4)*sqrt(z(2)^2 + z(4)^2)];
```

[†]this may not happen since Facebook seems to be diversifying in different directions.

Then, the solution of the differential equation (18) is obtained by the command:

```
[t,z] = ode45(f, tspan, z0);
```

where t_{span} specifies the integration time-span of the solution, for example, $t_{\text{span}} = [0, t_g]$ in the skydiver case. The vector z_0 specifies the initial values, for example, in the case of Eq. (11):

$$z_0 = \begin{bmatrix} 0 \\ v_0 \\ 0 \\ 0 \end{bmatrix} \quad (21)$$

The output column vector of times t is generated by the solution, and z is a matrix, such that its i th column, $z(:, i)$ is the solution for the i th variable $z(i)$, e.g., for the skydiver example, the column vectors $z(:, 1), z(:, 2), z(:, 3), z(:, 4)$ represent the solution vectors for $x(t), v_x(t), y(t), v_y(t)$ at the time vector t .

Often the differential equation to be solved has several additional parameters that need to be set by the user, such as the quantities $\{A_1, A_2, t_1, t_2, \rho, C, m, g\}$ of the skydiver example. There are basically three ways to handle such parameters. Let the parameters be denoted by p_1, p_2, \dots .

1. If the function $f(t, z)$ can be defined as a one-line anonymous function, then the parameters p_1, p_2, \dots , must be defined prior to defining the function $f(t, z)$, as we did in the above example:

```
% define the values of p1, p2, ..., here
% define f as anonymous function, parameters p1,p2,... may appear here

f = @(t,z) ... % define function handle and pass it to ode45
[t,z] = ode45(f, tspan, z0); % solution
```

2. Most likely the function and its parameter dependence cannot be defined as a single-line anonymous function. In this case one may define the function $f(t, z, p_1, p_2, \dots)$ in an M-file, say `f.m`, with a syntax

```
zdot = f(t,z,p1,p2,...)
```

Then, the parameters can be passed into **ode45** as follows:

```
% here f is the file name, and @f, the function handle

[t,z] = ode45(@f, tspan, z0, [], p1,p2,...);
```

where the `[]` argument is an empty options input and is placed there in order to allow the passing of the additional parameters p_1, p_2, \dots .

3. This is the simplest method. Suppose again that $f(t, z, p_1, p_2, \dots)$ is defined in an M-file. Then,

```
% define the values of p1, p2, ..., here
% define a new anonymous function g in terms of f and pass it into ode45

g = @(t,z) f(t,z,p1,p2,...); % g is a function handle
[t,z] = ode45(g, tspan, z0); % solution
```

The solutions to the skydiver problem illustrate all three methods. In that problem, the differential equation function M-file is called `fdot.m` and is listed below. It implements Eq. (20) and the time-variation of the area and drag constant of Eqs. (12) and (14), and has additional parameters $A_1, A_2, t_1, t_2, \rho, C, m, g$:

```
% -----
function zdot = fdot(t,z, A1,A2,t1,t2,rho,C,m,g)
```



```

tmid = (t1+t2)/2;
A = (A2-A1)/2 * tanh((t-tmid)/(t2-t1)*10) + (A2+A1)/2; % area at time t
%A = A1*(t<=tmid) + A2*(t>tmid); % enable for part (e)
D = rho*C*A/2/m; % drag coefficient
v = sqrt(z(2)^2 + z(4)^2); % velocity magnitude
zdot = [z(2); -D*z(2)*v; z(4); g - D*z(4)*v]; % differential equation
% -----

```

In the main program, **ode45** is called as follows to solve the differential equation:

```

% define the parameters A1,A2,t1,t2,rho,C,m,g, and tg and v0, here

    tspan = [0,tg]; % time span
    z0 = [0; v0; 0; 0]; % initial conditions

% using version-2:

    [t,z] = ode45(@fdot, tspan, z0, [], A1,A2,t1,t2,rho,C,m,g);

% or, version-3:

    gdot = @(t,z) fdot(t,z, A1,A2,t1,t2,rho,C,m,g); % new handle

    [t,z] = ode45(gdot, tspan, z0);

% extract x,vx,y,vy solutions from z and plot them vs. t:

    x = z(:,1); vx = z(:,2); y = z(:,3); vy = z(:,4);

    figure; plot(t,x); % etc.

```

Homework Problems – Week 9
440:127 – Spring 2015 – S. J. Orfanidis

- End-of-Chapter Problem 9.14. Please solve this problem using a for-loop with a break condition. In addition, please solve also the case when the error is required to be 10^{-12} .
- The *Fibonacci sequence* is generated by the following recursion, initialized at two arbitrary values f_1 and f_2 at $n = 1$ and $n = 2$, respectively:

$$f_n = f_{n-1} + f_{n-2}, \quad n \geq 3$$

The exact formula for the sequence is given by:

$$f_n = A \cdot \left(\frac{1 + \sqrt{5}}{2} \right)^{n-1} + B \cdot \left(\frac{1 - \sqrt{5}}{2} \right)^{n-1}, \quad n \geq 1$$

where A, B are given as follows in terms of the initial values f_1, f_2 :

$$A = \frac{(\sqrt{5} - 1)f_1 + 2f_2}{2\sqrt{5}}, \quad B = \frac{(\sqrt{5} + 1)f_1 - 2f_2}{2\sqrt{5}}$$

The sequence f_n is exponentially increasing.

- For the initial values $f_1 = 2$ and $f_2 = 4$, use a forever while-loop to determine the smallest index n , say n_0 , such that the sequence f_n will exceed the value $F = 10^6$, that is, n_0 is the smallest index such that $f_n \geq F$, for $n \geq n_0$, or, equivalently $f_n < F$, for $n < n_0$.
 Capture the results of the while-loop into an array f_n for $1 \leq n \leq n_0$. Then, use a single vectorized expression to calculate the sequence from the above formula, say g_n , and make a table comparing the results obtained with the while-loop, as the one shown at the end.
- Repeat part (a) using a conventional while-loop, but now use the values $f_1 = 5$, $f_2 = 2$, and $F = 10^7$.
- For the array f_n obtained in part (b), calculate the ratio of two successive Fibonacci numbers f_n/f_{n-1} , for $2 \leq n \leq n_0$, which is rapidly converging to the Golden ratio,

$$\phi = \frac{1 + \sqrt{5}}{2} = \text{Golden Ratio}$$

Plot the golden ratio error, $\left| \frac{f_n}{f_{n-1}} - \phi \right|$, versus n using a semilogy plot. See example graph at end.

- A famous problem in probability is the *birthday problem*: What is the minimum number of people such that the probability is more than 50% that at least two people will have the same birthday? The surprising answer is only 23 people, for which the probability is 50.7%. Clearly, if there are more than 365 people (ignoring leap years), the probability is 100%.

Consider a group of N people, with $N \leq 365$. The probability that no one in the group has the same birthday is:

$$P_{\text{none}} = \frac{(365 - 1)}{365} \cdot \frac{(365 - 2)}{365} \cdots \frac{(365 - N + 1)}{365} = \prod_{k=1}^{N-1} \frac{365 - k}{365} = \prod_{k=1}^{N-1} \left(1 - \frac{k}{365} \right)$$

This follows by noting that the probability that the second person has a different birthday from the first is $364/365$ (we are ignoring leap years), the probability that the third person has a different birthday from the first two is $363/365$, and so on. Thus, the probability that at least two persons will have the same birthday will be $1 - P_{\text{none}}$, or,

$$P(N) = 1 - \prod_{k=1}^{N-1} \left(1 - \frac{k}{365} \right), \quad 1 \leq N \leq 365 \quad (1)$$

- a. Write a one-line anonymous function **prob** that for a given N calculates $P(N)$. It must have syntax:

`P = prob(N);`

Then, write another version of the function **prob** (as an M-file) that uses a for-loop to construct $P(N)$.

- b. Using the above function **prob** and a while-loop determine the smallest N for which $P(N) \geq 0.50$. Find also the smallest N such that $P(N) \geq 0.90$, and the N such that $P(N) \geq 0.99$.
- c. Using the function **prob** and a for-loop, calculate all the probabilities $P(n)$ for $1 \leq n \leq 100$, and plot them versus n . Indicate the three special points of part (b) on the graph.
- d. A simple analytical approximation to Eq. (1) is given by:

$$F(N) = 1 - \exp\left(-\frac{N(N-1)}{730}\right), \quad 1 \leq N \leq 365 \quad (2)$$

Repeat the plot of part (c) and to it add the values obtained from the approximation of Eq. (2).

4. *Savings Account.* Suppose you deposit a certain amount of x_0 dollars a month for M months in a savings account that offers an annual rate of R percent, so that the monthly rate is $r = R/1200$. Assume that the initial deposit in opening the account was y_0 dollars. After the period of M months, you begin withdrawing a certain amount of x dollars per month while earning the same interest. How long will your account last before it is depleted to zero? Can it last forever?

Let $y(n)$ be the balance of the account at the end of the n th month. It satisfies the difference equation:

$$y(n) = \begin{cases} y(n-1) + ry(n-1) + x_0, & \text{if } 1 < n \leq M \\ y(n-1) + ry(n-1) - x, & \text{if } n > M \end{cases} \quad (3)$$

and initialized at $y(1) = y_0$. Define the quantity $x_{\min} = ry(M)$. It can be shown that if the withdrawal amount is $x > x_{\min}$, then the account will be depleted; otherwise, it will last forever. In the former case, one can calculate analytically the number N of additional months that it will take to deplete the account by solving the equation:

$$(1+r)^N = \frac{x}{x - x_{\min}} \quad (4)$$

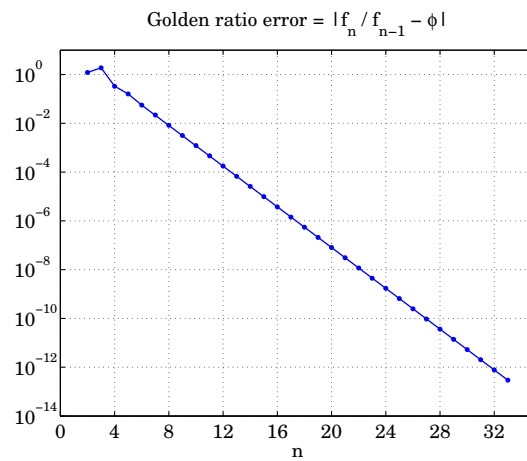
For the rest of this problem assume the following values:

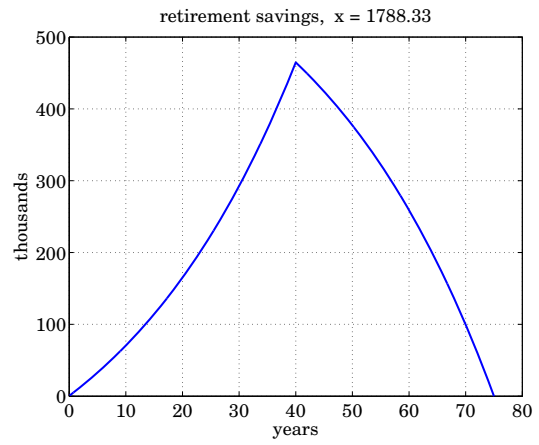
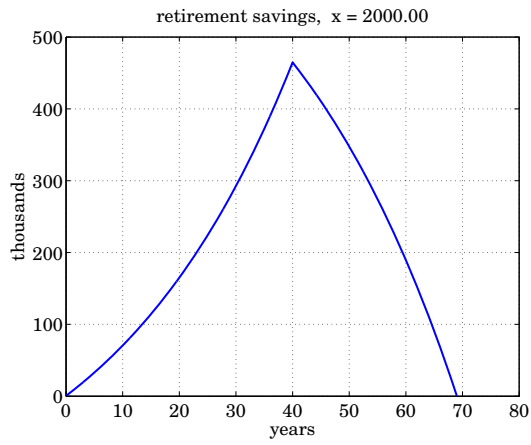
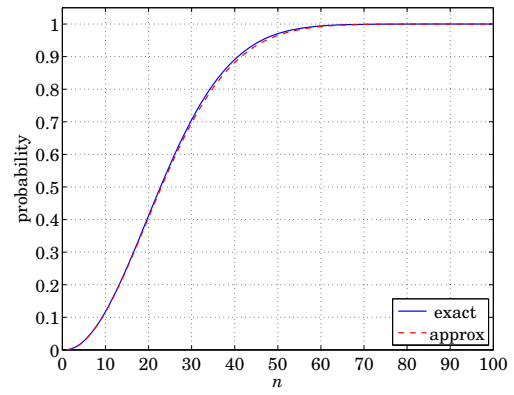
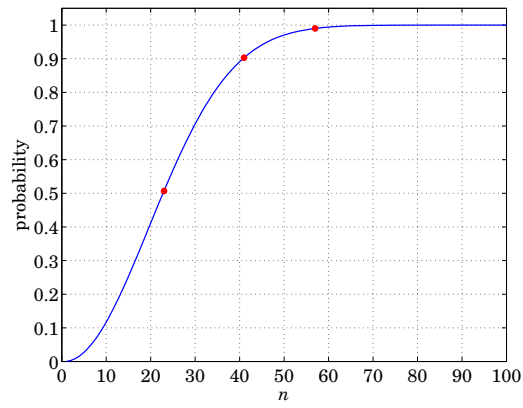
$$\begin{aligned} R &= 3\% \text{ annual} \\ y_0 &= \$1000 \\ x_0 &= \$500, \quad x = \$2000 \\ M &= 12 \times (40 \text{ years}) = 480 \text{ months} \end{aligned}$$

- a. Using a for-loop, run the difference equation (3) for $1 < n \leq M$ to determine $y(M)$ and then calculate x_{\min} . Verify that $x > x_{\min}$ so that the account will be depleted.
- b. Using a while-loop, iterate Eq. (3) for $n > M$ for as long as $y(n)$ remains nonnegative, that is, as long as $y(n) \geq 0$. Determine the number N of iterations until this condition is violated—that's when your account will be depleted.
Express N in years. Solve Eq. (4) for N and verify that it agrees with the one calculated using the while-loop.
Plot $y(n)$ versus n for the entire life of the account, $1 \leq n \leq N + M$, but using units of years as your x -axis, and thousand-dollars as your y -axis.
- c. Suppose you wish your account to last 35 years beyond the initial M , i.e., $N = 12 \times 35 = 420$ months. Using Eq. (4), determine the amount x that you should be withdrawing every month for the last N months. Then, using a for-loop, calculate and plot $y(n)$ versus n for the entire life of the account, that is, for $1 \leq n \leq N + M$.

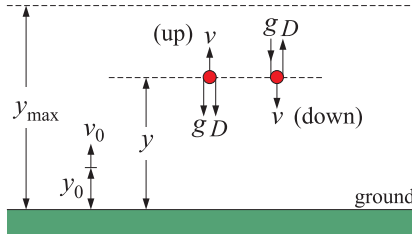
Typical Tables and Graphs for Problems 1-4

n	f(n)	g(n)
1	2	2
2	4	4
3	6	6
4	10	10
5	16	16
6	26	26
7	42	42
8	68	68
9	110	110
10	178	178
11	288	288
12	466	466
13	754	754
14	1220	1220
15	1974	1974
16	3194	3194
17	5168	5168
18	8362	8362
19	13530	13530
20	21892	21892
21	35422	35422
22	57314	57314
23	92736	92736
24	150050	150050
25	242786	242786
26	392836	392836
27	635622	635622
28	1028458	1028458





5. *Air Drag on a Rising and Falling Ball.* A ball is thrown vertically upwards at time $t = 0$ with some initial velocity v_0 from an initial height y_0 . As the ball rises, gravity and air drag act downwards slowing down the ball. The ball reaches a maximum height, say, y_{\max} at time t_{\max} . Then, it begins to fall down but now the air drag is acting upwards slowing down the ball until it reaches a terminal velocity, say, v_c , and eventually the ball hits the ground at some time, say, t_g . The figure below depicts the directions of the forces on the rising and falling ball. Assuming that the air drag is quadratic in the velocity, the acceleration of the rising and falling ball is given by Eq. (5) below.



$$a = \frac{dv}{dt} = -g - D \cdot v \cdot |v| = -g - D \cdot v^2 \cdot \text{sign}(v) = \begin{cases} -g - D v^2, & v \geq 0, \text{ rising} \\ -g + D v^2, & v \leq 0, \text{ falling} \end{cases} \quad (5)$$

where we note, $|v| = v \cdot \text{sign}(v)$, and g is the acceleration of gravity and D is an air drag constant related to the mass m , air density ρ_a , the ball's cross-sectional area A , and drag coefficient C , by

$$D = \frac{\rho_a C A}{2m}$$

Eq. (5) ignores the buoyancy force, but that can be easily taken into account by simply replacing g by a new effective g as follows, where ρ is the density of the ball:

$$g_{\text{eff}} = g \left(1 - \frac{\rho_a}{\rho} \right)$$

In this problem, we will assume a very dense ball $\rho \gg \rho_a$, so that the buoyancy force is negligible. On the other hand, for a hot-air balloon, we have $\rho < \rho_a$, and g_{eff} will reverse sign causing a lift.

The objective of this problem is to replace Eq. (5) by a difference equation that can be solved numerically in MATLAB using a while-loop, and to determine the height $y(t)$ as a function of time, as well as the maximum height and time y_{\max} , t_{\max} , and the ground time t_g . Moreover, since this problem can be solved analytically, we will compare the numerical and analytical solutions. Let us define the following parameters in terms of g, D , where v_c is the terminal velocity:

$$v_c = \sqrt{\frac{g}{D}}, \quad t_c = \frac{v_c}{g} = \frac{1}{\sqrt{gD}}, \quad h_c = v_c t_c = \frac{v_c^2}{g} = \frac{1}{D} \quad (6)$$

Then, the time t_{\max} to reach the maximum height is given in terms of the initial velocity v_0 by:

$$v_0 = v_c \cdot \tan\left(\frac{t_{\max}}{t_c}\right) \Rightarrow t_{\max} = t_c \cdot \text{atan}\left(\frac{v_0}{v_c}\right) \quad (7)$$

The solution of Eq. (5) is given as follows, where the time interval $0 \leq t \leq t_{\max}$ represents the rising period, and $t \geq t_{\max}$, the falling period:

$$v(t) = \begin{cases} v_c \cdot \tan\left(\frac{t_{\max} - t}{t_c}\right), & 0 \leq t \leq t_{\max} \\ v_c \cdot \tanh\left(\frac{t_{\max} - t}{t_c}\right), & t \geq t_{\max} \end{cases} \quad (8)$$

Integrating the equation $dy/dt = v$, we determine the height $y(t)$ as measured from the ground up:

$$y(t) = \begin{cases} y_{\max} + h_c \cdot \ln \left[\cos \left(\frac{t_{\max} - t}{t_c} \right) \right], & 0 \leq t \leq t_{\max} \\ y_{\max} - h_c \cdot \ln \left[\cosh \left(\frac{t_{\max} - t}{t_c} \right) \right], & t \geq t_{\max} \end{cases} \quad (9)$$

where y_{\max} is given as follows in terms of the initial height y_0 :

$$y_{\max} = y_0 - h_c \cdot \ln \left[\cos \left(\frac{t_{\max}}{t_c} \right) \right] \quad (10)$$

Requiring that $y = 0$ at $t = t_g$, results in the following expression for t_g :

$$y_{\max} - h_c \cdot \ln \left[\cosh \left(\frac{t_{\max} - t_g}{t_c} \right) \right] = 0 \quad \Rightarrow \quad t_g = t_{\max} + t_c \cdot \operatorname{acosh} \left[\exp \left(\frac{y_{\max}}{h_c} \right) \right] \quad (11)$$

Now to the numerical solutions. We discretize the time by $t_n = (n - 1)T$, for $n = 1, 2, 3, \dots$, where T is a very small time step. Then, the differential equations,

$$\frac{dv}{dt} = a = -g - D \cdot v \cdot |v|, \quad \frac{dy}{dt} = v$$

can be replaced by the difference equations:

$$\frac{v(n+1) - v(n)}{T} = a(n) = -g - D \cdot v(n) \cdot |v(n)|, \quad \frac{y(n+1) - y(n)}{T} = v(n) \quad (12)$$

where $a(n), v(n), y(n)$ denote the numerical approximations to $a(t_n), v(t_n), y(t_n)$, respectively. Rewriting Eq. (12), we obtain our final numerical computation algorithm that calculates $a(n), v(n), y(n)$:

```

initialize:
    v(1) = v_0,  y(1) = y_0
for n = 1, 2, 3, ..., do:
    a(n) = -g - D * v(n) * |v(n)|
    v(n+1) = v(n) + a(n) * T
    y(n+1) = y(n) + v(n) * T

```

(13)

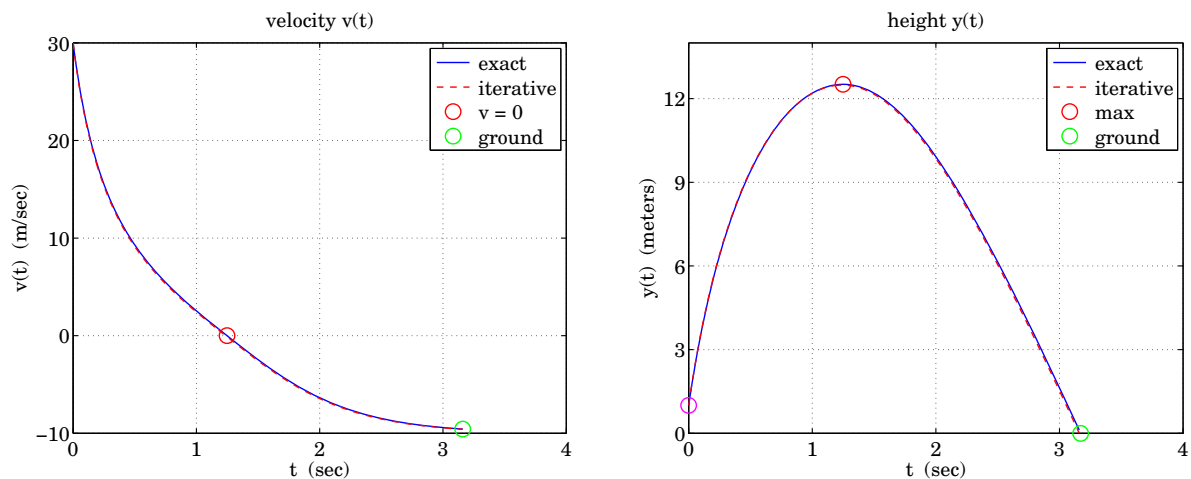
We may also consider the following slightly different versions that often work better (see References at end), however, the results obtained with Eq. (13) are perfectly adequate for this problem:

<pre> initialize: v(1) = v_0, y(1) = y_0 for n = 1, 2, 3, ..., do: a(n) = -g - D * v(n) * v(n) v(n+1) = v(n) + a(n) * T y(n+1) = y(n) + v(n+1) * T </pre>	<pre> initialize: v(1) = v_0, y(1) = y_0 for n = 1, 2, 3, ..., do: a(n) = -g - D * v(n) * v(n) v(n+1) = v(n) + a(n) * T y(n+1) = y(n) + v(n) * T + 1/2 * a(n) * T^2 </pre>
--	---

In the rest of the problem, consider the following numerical values (we rounded g for convenience):

$$g = 10 \text{ m/sec}^2, \quad D = 0.1 \text{ m}^{-1}, \quad v_0 = 30 \text{ m/sec}, \quad y_0 = 1 \text{ m}, \quad T = 0.01 \text{ sec}$$

- a. Using a *forever while-loop*, iterate Eq. (13) until ground is reached, that is, insert a condition to exit the loop when $y(n)$ becomes negative. Determine the last iteration index, say, n_g , just before ground is reached, and calculate the corresponding time in seconds using $t_g = (n_g - 1)T$.
Moreover, insert some code in the loop to determine the maximum height reached, y_{\max} , and the time index, n_{\max} , and actual time $t_{\max} = (n_{\max} - 1)T$, when that maximum is reached.
- b. Construct the vector of times $t(n) = (n - 1)T$, for $n = 1 : n_g$, and plot $v(n)$ and $y(n)$ versus t . Indicate the points y_0, y_{\max} , and $y = 0$ on the graphs.
Moreover, add to the graphs the corresponding plots of $v(t)$ and $y(t)$ obtained from the exact analytical expressions given above.
- c. Repeat parts (a,b) with the larger $T = 0.02$ sec, which will make more visible the slight discrepancy between the iterative and analytical solutions.



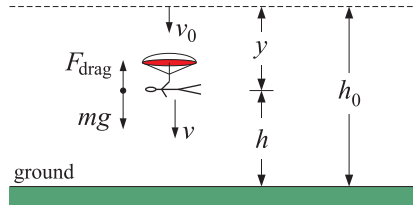
References - PDFs on Sakai

1. A. Cromer, "Stable Solutions Using the Euler Approximation," *Am. J. Phys.*, **49**, 455 (1981).
2. R. W. Stanley, "Numerical Methods in Mechanics," *Am. J. Phys.*, **52**, 499 (1984).
3. A. D'Innocenzo and L. Renna, "Analysis of Some Elementary Numerical Methods in Mechanics," *Eur. J. Phys.*, **13**, 153 (1992).
4. I. R. Gatland, "Numerical Integration of Newton's Equations Including Velocity-Dependent Forces," *Am. J. Phys.*, **62**, 259 (1996).
5. T. Timberlake and J. E. Hasbun, "Computation in Classical Mechanics," *Am. J. Phys.*, **76**, 334 (2008).

6. *Supersonic Free Fall.* In October 2012, Felix Baumgartner jumped off a balloon from the stratospheric altitude of 39 km (above sea-level), reaching after 50 sec a supersonic speed of 377 m/sec, at an altitude of 28 km. The speed of sound at that altitude is 300 m/sec, therefore, he achieved a record speed of Mach 1.25. After falling for 260 seconds and reaching an altitude of 2.5 km, he opened the parachute, and eventually reached the ground after a total falling time of 9 minutes and 18 sec. Note that the ground at the landing site in Roswell, New Mexico was at an elevation of 1043 meters above sea level. The following data were recorded during the fall, see Ref. [1] at the end.

t (sec)	v (m/s)	h (km)	v_{sound} (m/s)	
0	0	38.969	315.38	jump altitude above sea level
34	309.72	33.446	305.48	begin supersonic speeds
50	377.11	27.833	300.28	maximum supersonic speed
64	289.72	22.960	297.02	end supersonic speeds
180	79.17	7.619	309.71	speed slows substantially
260	53.19	2.567	330.30	parachute opens
558	-	1.043	336.27	ground is reached

The purpose of this problem is to reproduce these results with MATLAB. We recall from the previous Problem that the air-drag force and downward vertical force are given by:



$$F_{\text{drag}} = \frac{1}{2} \rho C A v^2$$

$$m \frac{dv}{dt} = mg - F_{\text{drag}} = mg - \frac{1}{2} \rho C A v^2$$

which may be re-written in the form:

$$\frac{dv}{dt} = g \cdot \left(1 - \frac{v^2}{v_c^2} \right), \quad v_c = \sqrt{\frac{2mg}{CA\rho}} = \text{terminal velocity} \quad (14)$$

At high altitudes h , both the acceleration of gravity g and the air density ρ , and hence v_c , depend on the height h , and can be represented by the following functions:

$$g(h) = \frac{g_0 R_e^2}{(R_e + h)^2}, \quad g_0 = 9.80665 \text{ m/s}^2, \quad R_e = 6356.766 \text{ km}$$

$$\rho(h) = 1.2241 \cdot \exp \left[- \left(\frac{h}{11.661} \right) - \left(\frac{h}{18.192} \right)^2 + \left(\frac{h}{29.235} \right)^3 \right], \quad 0 \leq h \leq 40 \text{ km} \quad (15)$$

$$v_c(h) = \sqrt{\frac{2mg(h)}{CA\rho(h)}} = \text{height-dependent terminal velocity}$$

where h is in units of km in all expressions, and R_e is the radius of the earth. The function $\rho(h)$ provides a simple and accurate least-squares fit to the standard atmosphere data over the interval $0 \leq h \leq 40$ km, and will be derived in a future homework set. Thus, Eq. (14) must be replaced by:

$$\frac{dv}{dt} = g(h) \cdot \left(1 - \frac{v^2}{v_c^2(h)} \right) = \text{acceleration} \quad (16)$$

$$\frac{dh}{dt} = -v = \text{speed}$$

where h is measured upwards from the ground and is related to the drop-distance y from the initial height by $h = h_0 - y$, as shown in the above figure. The negative sign in the second equation is because v represents the downward velocity and dh/dt , the upward velocity.

Next, we discretize the time in small time steps $t_n = (n - 1)T$, $n = 1, 2, \dots$, and replace the time-derivatives by differences to obtain the computational algorithm:

$$\frac{v(n+1) - v(n)}{T} = a(n) = g(h(n)) \cdot \left(1 - \frac{v^2(n)}{v_c^2(h(n))}\right)$$

$$\frac{h(n+1) - h(n)}{T} = -v(n)$$

which can be rearranged as follows, where we also divided the velocity term of $h(n)$ by 1000 because $h(n)$ is in km instead of meters:

<pre> initialize at: h(1) = h0, v(1) = v0 = 0 for n = 1, 2, 3, ..., do: a(n) = g(h(n)) * (1 - (v^2(n) / (v_c^2(h(n)))) v(n+1) = v(n) + a(n) * T h(n+1) = h(n) - v(n) * T / 1000 </pre>	(17)
--	------

In the rest of this problem assume the following parameter values:

$m = 118$ kg	260 lbs, Baumgartner's total weight including equipment, Ref. [1]
$C_1 = 1.0$	drag coefficient during free-fall
$A_1 = 0.7$ m ²	cross-sectional area during free-fall
$C_2 = 3.17$	drag coefficient after parachute opens
$A_2 = 25.1$ m ²	parachute area, 270 ft ² , Ref. [1]
$v_0 = 0$	initial jump velocity
$h_0 = 38.969$ km	initial jump altitude, Ref. [1]
$h_s = 1.043$ km	landing site elevation above sea level, Ref. [1]
$t_{260} = 260$ sec	time parachute opens, Ref. [1]
$T = 0.01$ sec	time step

For the parameters not found in Ref. [1], we chose reasonable values to see if the above model can adequately describe the fall. The value of C_2 seems excessive but we estimated it in the following way. The parachute was opened at a distance of $(2567 - 1043) = 1524$ meters above ground, and it took $(558 - 260) = 298$ sec to land. Therefore, an approximate estimate of the terminal velocity with the parachute open is $v_c = 1524/298 = 5.1$ m/sec. Since the area of the parachute is known, one can solve the equation $v_c = \sqrt{2mg/\rho C A}$ for C , which gives $C = 3.17$ using the values of ρ, g at the altitude of $h_s = 1.043$ km.

- a. Write a MATLAB function that calculates the speed of sound in air at any vector of heights h , as well as the corresponding absolute temperatures:

$$[vs, T] = \text{vsound}(h)$$

where h should be in km. Use the following formula for it:

$$v_s(h) = \sqrt{\gamma R_s T(h)}$$

where $\gamma = 1.4$ is the adiabatic expansion constant of air, $R_s = 287.053$ is the specific gas constant of air in J/K/kg, and $T(h)$ is the absolute temperature (K) as a function of height as defined in the Standard Atmosphere Problem of Week-7. Plot the sound speed $v_s(h)$ and temperatures $T(h)$ versus h in the range $0 \leq h \leq 40$ km.

Define single-line anonymous MATLAB functions implementing $g(h)$ and $\rho(h)$ from Eq. (15). Define a MATLAB function $v_c(h, CA)$ to be used with the two values of the coefficients C_1A_1 and C_2A_2 :

$$v_c(h, CA) = \sqrt{\frac{2mg(h)}{CA\rho(h)}} \quad (18)$$

where CA is thought of as a single variable. Plot $g(h)$, $\rho(h)$, and $v_c(h, C_1A_1)$, $v_c(h, C_2A_2)$, for $0 \leq h \leq 40$ km.

- b. Run the following forever while-loop that implements Eq. (17), with the added feature that it uses $v_c(h, C_1A_1)$ while the parachute is closed and then switches to $v_c(h, C_2A_2)$ when the parachute opens at time $t = 260$ sec. The loop constructs the vectors t, v, h .

```

h(1) = h0;
v(1) = v0;
n = 1;

while 1

    if h(n)<hs, break; end           % break if ground altitude is reached

    t(n) = (n-1)*T;

    H = h(n);                       % altitude at time t(n)

    CA = C1*A1*(t(n)<=t260) + C2*A2*(t(n)>t260); % parachute closed
                                           % parachute opens after t=t260

    a(n) = g(H)*(1 - v(n)^2/vc(H,CA)^2);

    v(n+1) = v(n) + a(n)*T;          % v(n) in m/s, a(n) in m/s^2, T in sec

    h(n+1) = h(n) - v(n)*T/1000;     % h in units of km

    n = n+1;

end

```

- c. From the exit condition of the loop, determine the total time to reach ground, t_g , in seconds. For the purpose of comparing the calculated values to the observed ones given above, calculate also the following data points.

Using the function **max**, determine the maximum velocity reached, v_{\max} , and the time instant t_{\max} and height h_{\max} at which it is reached.

Determine also the time, velocity, and height, say, t_b, v_b, h_b , when the fall becomes supersonic. And also the time, velocity, and height, say, t_e, v_e, h_e , when the supersonic speeds end and they become subsonic.

Determine the velocity and height corresponding to the time $t = 180$ sec, as well as those corresponding to $t = 260$ sec when the parachute opens.

Using at most seven **fprintf** commands print the given data, and the above calculated data points, exactly as shown below.

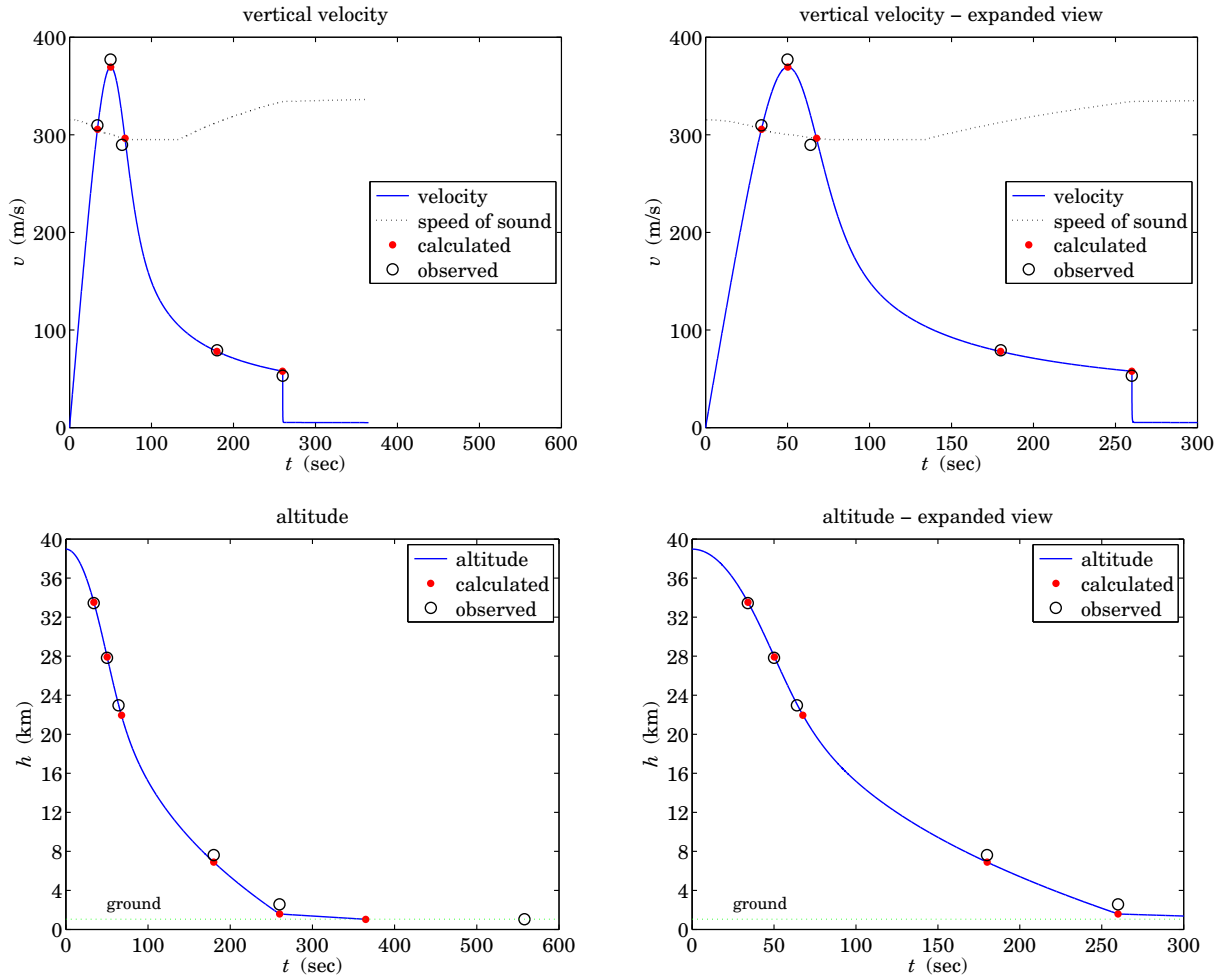
notes	observed			calculated		
	t (sec)	v (m/s)	h (km)	t (sec)	v (m/s)	h (km)
initial height	0	0.00	38.969	0.00	0.00	38.969
begin supersonic	34	309.72	33.446	34.07	305.57	33.531
maximum supersonic speed	50	377.11	27.833	50.23	369.39	27.925
end supersonic	64	289.72	22.960	67.65	296.35	21.950
speed slows substantially	180	79.17	7.619	180.00	77.94	6.901
parachute opens	260	53.19	2.567	260.00	57.75	1.589
ground	558	-	1.043	364.65	5.11	1.043

- d. Plot the calculated speed $v(t)$ and height $h(t)$ versus time t over the interval $0 \leq t \leq t_g$, and indicate on the graphs the observed and calculated data points from part (c), as shown below.

We note that the above model describes the overall motion fairly accurately. Our calculated altitude corresponding to $t = 260$ sec is shorter than the observed one, and this causes the total landing time to be shorter. A possible explanation is that after reaching subsonic speeds around $t = 64$ sec, Baumgartner went into a period of spinning and changing orientation before stabilizing again. Our simplified model did not take that into account and this could have increased the drag coefficient C_1 and surface area A_1 for a period of time, resulting in the observed higher altitude at $t = 260$. Nevertheless, the model confirms the main features of free fall in an atmosphere with density that diminishes with altitude, namely, that as the fall proceeds, the speed first increases to a maximum, and then decreases to a terminal velocity, but not quite achieving it, see Refs. [5-7].

References - PDFs on Sakai

1. Red Bull Stratos Project, 2012.
<http://www.redbullstratos.com/>
<http://www.redbullstratos.com/science/scientific-data-review/>
http://issuu.com/redbullstratos/docs/red_bull_stratos_factsheet_final_statistics_050213
2. F. R. Greening, "Baumgartner's Jump and the Physics of Freefall," *Phys. Educ.*, **48**, 139 (2013).
3. F. Theilmann and M. Apolin, "Supersonic Freefall—A Modern Adventure as a Topic for the Physics Class," *Phys. Educ.*, **48**, 150 (2013).
4. A. W. Robinson and C. G. Patrick, "The Physics of Colonel Kittinger's Longest Lonely Leap," *Phys. Educ.*, **43**, 477 (2008).
5. J. Benacka, "High-Altitude Free Fall Revised," *Am. J. Phys.*, **78**, 616 (2010).
6. P. Mohazzabi and J. H. Shea, "High-Altitude Free Fall," *Am. J. Phys.*, **64**, 1242 (1996).
7. N. M. Shea, "Terminal Speed and Atmospheric Density," *Phys. Teacher*, **31**, 176 (1993).



7. *Air Drag on a Baseball.*[†] The concept of the drag coefficient was discussed in Example 2.3 of the text. The force due to the air resistance acting against a moving object is taken to be proportional to the square of the object's velocity, typically at speeds less than 200 mph:

$$F = \frac{1}{2} C \rho A v^2$$

where C, ρ, A are the drag coefficient, air density, and object's cross sectional area. The direction of the force is opposite of the velocity. In this problem, we will study the impact of air drag on the motion of a baseball. We will obtain the ball's trajectory using a discretized version of Newton's equations of motion and solve them using for-loops in MATLAB. For a ball of radius R , cross-sectional area $A = \pi R^2$, and mass m , let us define the normalized drag coefficient:

$$D = \frac{C \rho A}{2m} = \frac{C \rho \pi R^2}{2m}$$

Then, Newton's equations of motion take the following form for the x, y (horizontal and vertical) components of positions, velocities, and accelerations:

$$\begin{aligned} \frac{dx}{dt} &= v_x, & \frac{dv_x}{dt} &= a_x = -D v_x v, & v &= \sqrt{v_x^2 + v_y^2} \\ \frac{dy}{dt} &= v_y, & \frac{dv_y}{dt} &= a_y = -D v_y v - g \end{aligned} \tag{19}$$

[†]<http://wps.aw.com/wps/media/objects/877/898586/topics/topic01.pdf>

where g is the acceleration of gravity acting vertically downwards. These differential equations may be solved numerically using, for example, MATLAB's built-in differential equation solver `ode45`. However, in this problem, we are going to replace them with a discrete-time version that can very accurately determine the solution. We assume that time is discretized in small steps $t_n = nT$, $n = 0, 1, 2, \dots$, where T is a very small step increment. Let us denote by $x(n)$ the value of the horizontal distance $x(t_n)$ at time $t = t_n$, and similarly for the quantities $y(n)$, $v_x(n)$, $v_y(n)$. Then, Eq. (19) can be replaced by the following discretized version:[†]

$$\begin{aligned} v(n) &= \sqrt{v_x^2(n) + v_y^2(n)} \\ a_x(n) &= -D v_x(n) v(n) \\ a_y(n) &= -D v_y(n) v(n) - g \end{aligned} \quad (20)$$

$$\begin{aligned} x(n+1) &= x(n) + T v_x(n) + \frac{1}{2} T^2 a_x(n) \\ v_x(n+1) &= v_x(n) + T a_x(n) \\ y(n+1) &= y(n) + T v_y(n) + \frac{1}{2} T^2 a_y(n) \\ v_y(n+1) &= v_y(n) + T a_y(n) \end{aligned} \quad (21)$$

At time n , we assume that we know the quantities $x(n)$, $y(n)$, $v_x(n)$, $v_y(n)$. From Eq. (20), we calculate the accelerations $a_x(n)$, $a_y(n)$, and use them in Eq. (21) to calculate the next values $x(n+1)$, $y(n+1)$, $v_x(n+1)$, $v_y(n+1)$, and the process is repeated. To get the recursions started we may take the initial values to be:

$$x(0) = 0, \quad y(0) = h, \quad v_x(0) = v_0 \cos \theta_0, \quad v_y(0) = v_0 \sin \theta_0 \quad (22)$$

where h is the height of the initial launch, and v_0 , θ_0 , the initial velocity and angle of departure of the ball. The objective of this problem is to use a for-loop to carry out the iteration (21) and compare the resulting trajectory with the one obtained when the drag force is ignored. The latter is given by (see week-4 homework):

$$x = v_0 \cos \theta_0 t, \quad y = h + v_0 \sin \theta_0 t - \frac{1}{2} g t^2 \quad (23)$$

The maximum travel time, i.e., the time it takes to hit ground, is:

$$T_{\max} = \frac{v_0 \sin \theta_0}{g} + \sqrt{\frac{2h}{g} + \frac{v_0^2 \sin^2 \theta_0}{g^2}} \quad (24)$$

Choose an initial ball speed of $v_0 = 90$ mph and launch angle $\theta_0 = 30^\circ$, convert them to m/sec and radians, and use the following values for the other parameters (given in metric units):

$$h = 1, \quad C = 0.5, \quad R = 0.0366, \quad \rho = 1.2, \quad m = 0.145, \quad g = 9.81$$

- Choose a sampling time interval of $T = 1/100$ sec, calculate T_{\max} in seconds, and define the maximum number of samples to use in the iterative algorithm by $N = \text{floor}(T_{\max}/T)$. Define the time vector $t = 0 : T : T_{\max}$ and compute the corresponding x, y vectors in the no-drag case from Eq. (24).
- Use a for-loop to iterate Eqs. (20) and (21) for $n = 0 : N$ and calculate the arrays $x(n)$, $y(n)$, $v_x(n)$, $v_y(n)$ (note that n in these expressions is the time index and you will need to shift it by 1 to make it into a MATLAB index.) Because the air drag force slows down the motion, the ball will hit the ground faster than in the no-drag case. Put some conditional code in your for-loop that breaks out of the loop when the ball hits ground. Therefore, the effective length of the resulting arrays $x(n)$, $y(n)$ will be shorter than N . Determine the value of n , say n_{\max} , and the corresponding travel time $t_{\max} = n_{\max}T$ when the ball hits ground.

[†] this is obtained by assuming that the accelerations remain approximately constant within each small time interval $[t_n, t_{n+1}]$.

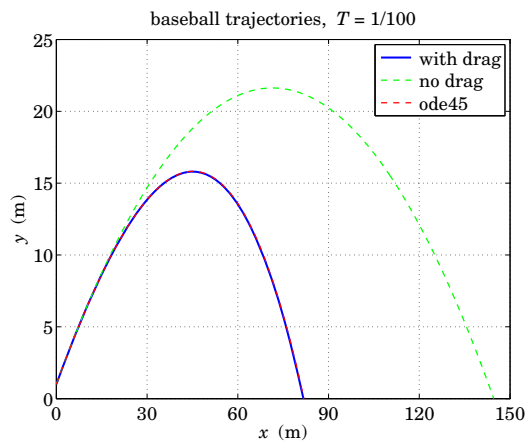
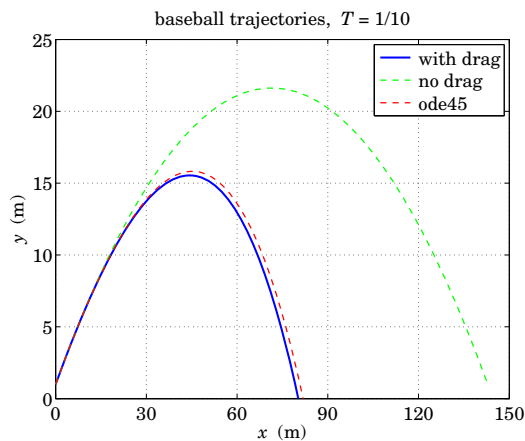
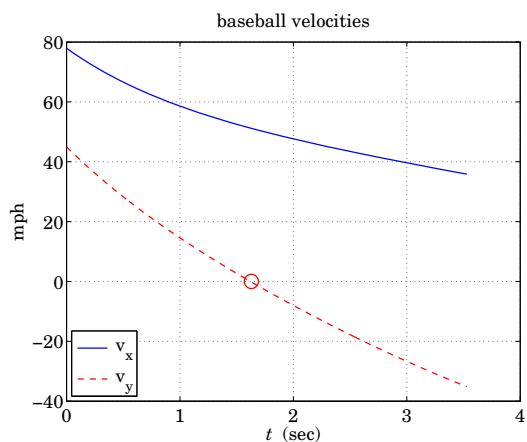
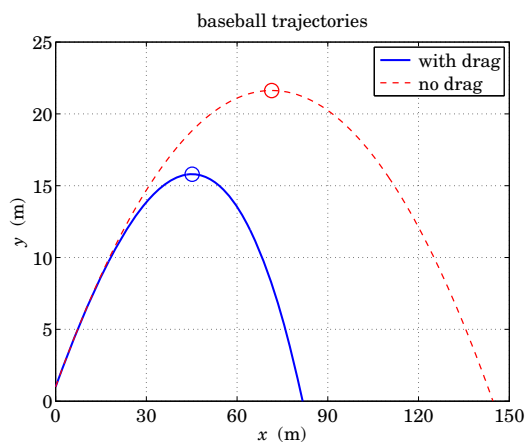
- c. On the same graph, plot the trajectories (y vs. x) that you computed in parts (a,b). Determine and place on the curves the points (x_{\max}, y_{\max}) that correspond to the maximum heights of both trajectories. For the no-drag case, these points are determined by the formulas:

$$x_{\max} = \frac{v_0^2 \cos \theta_0 \sin \theta_0}{g}, \quad y_{\max} = h + \frac{v_0^2 \sin^2 \theta_0}{2g}$$

For both cases, determine the time t_0 it takes to reach the maximum height.

- d. For the air-drag case, plot on the same graph the velocity vectors v_x, v_y versus time over the interval $0 \leq t \leq t_{\max}$, e.g., you may choose $t = \text{linspace}(0, t_{\max}, n_{\max})$. Place on the graph the point $(t_0, v_y = 0)$ that corresponds to the ball reaching its maximum height.

Note: The solution of Eq. (19) using the **ode45** solver is almost indistinguishable from the above solution, as long as T is small enough. The homework solutions will illustrate how to use **ode45** function - see the file **ode45ex.m**. The bottom two graphs demonstrate the **ode45** method for the two choices of $T = 1/10$ and $T = 1/100$.



Homework Problems - Week 10
440:127 - Spring 2015 - S. J. Orfanidis

1. End-of-Chapter Problems 10.18 and 10.21.
2. *Jacobi Method*. Consider the following linear system:

$$\begin{aligned} 5x_1 + 2x_2 + x_3 + x_4 &= 10 \\ 2x_1 + 6x_2 + 2x_3 + x_4 &= 12 \\ x_1 + 2x_2 + 7x_3 + 2x_4 &= -8 \\ x_1 + x_2 + 2x_3 + 8x_4 &= 29 \end{aligned}$$

- a. Solve this system by casting it in a matrix form and using the backslash operator.
- b. Solve the system using the Jacobi iterative method as outlined in the week-10 lecture notes. Use an error tolerance of 10^{-12} and a zero initial vector. Determine the iterative solution and the number of iterations k it takes to converge. Calculate the (norm of the) difference between the iterative solution and that obtained in part (a). Plot the four components of the iterated vector $\mathbf{x}(k)$ versus iteration number k , as shown below.
- c. Repeat parts (a,b) for the following system:

$$\begin{aligned} 3x_1 - 2x_2 + 7x_3 &= 20 \\ x_1 + 6x_2 - x_3 &= 10 \\ 10x_1 - 2x_2 + 7x_3 &= 27 \end{aligned}$$

For part (b), use the initial vector $\mathbf{x}_0 = \begin{bmatrix} -1 \\ 0 \\ 4 \end{bmatrix}$.

Hint: The system may require some rearrangement.

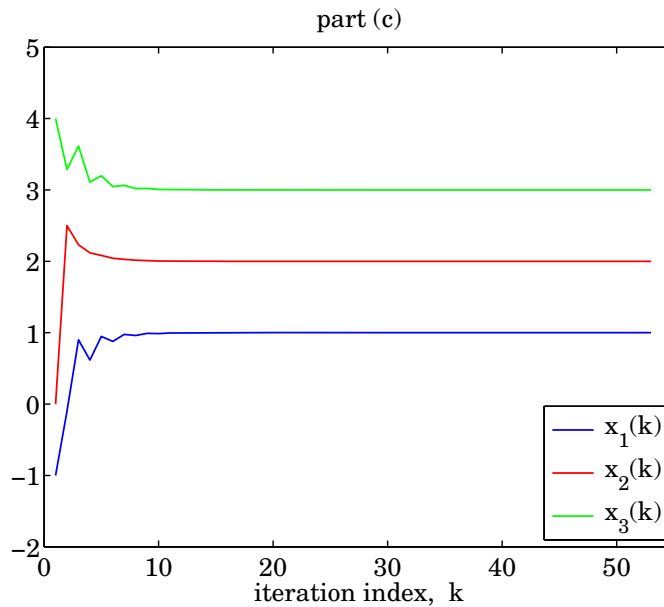
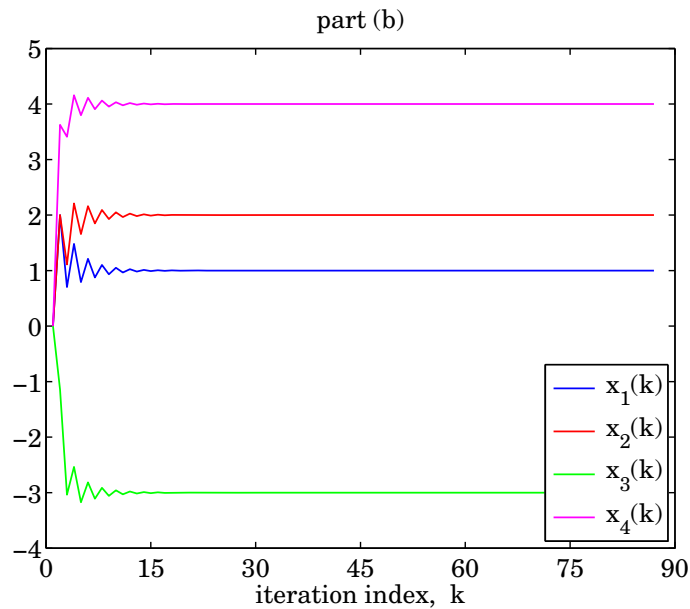
- d. Write a MATLAB function called **jacobi** that implements the Jacobi iterative method for solving a linear system $\mathbf{Ax} = \mathbf{b}$. It should have syntax:

```
[x,k,X,B,c,rho] = jacobi(A,b,tol,x0);

% A = NxN coefficient matrix in Ax=b
% b = Nx1 right-hand-side in Ax=b
% tol = error tolerance
% x0 = Nx1 initial vector, default x0 = zeros(N,1)

% x = Nx1 converged vector
% k = number of iterations to converge
% X = Nxk matrix whose columns are the successive iterated vectors
% B = Jacobi iteration matrix
% c = transformed vector b
% rho = spectral radius of Jacobi iteration matrix
```

The function should check if the method is applicable and produce a message and exit if it is not. Also if the fourth argument, \mathbf{x}_0 , is omitted, the initial vector should be set to zero by default. Verify your function on the above examples.



3. A *stochastic matrix* is a square matrix of non-negative entries such that each column adds up to 1. Such matrices appear in *Markov chain* models and have a wide range of applications in engineering, science, biology, economics, and internet search engines, such as Google's pagerank matrix (which has size in the billions.) Given such a matrix P whose entries are strictly positive, then there is a theorem that guarantees the existence of a steady-state equilibrium vector \mathbf{x} such that $\mathbf{x} = P\mathbf{x}$. Moreover, this vector can be computed recursively starting from an arbitrary initial vector \mathbf{x}_0 by the recursion:

$$\mathbf{x}_{k+1} = P\mathbf{x}_k, \quad k = 0, 1, 2, \dots \quad (1)$$

and \mathbf{x}_k converges to \mathbf{x} as $k \rightarrow \infty$, regardless of the initial vector \mathbf{x}_0 . The solution of Eq. (1) can be given explicitly as the matrix operation:

$$\mathbf{x}_k = P^k \mathbf{x}_0, \quad k = 0, 1, 2, \dots \quad (2)$$

Consider the following example:

$$P = \begin{bmatrix} 0.5 & 0.2 & 0.3 \\ 0.3 & 0.7 & 0.3 \\ 0.2 & 0.1 & 0.4 \end{bmatrix}, \quad \mathbf{x}_0 = \begin{bmatrix} 0.1 \\ 0.3 \\ 0.6 \end{bmatrix} \quad (3)$$

a. Denote the equilibrium vector by

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

To make it unique, we will assume that its entries add up to 1, that is, $x_1 + x_2 + x_3 = 1$. Set up three equations in the three unknowns $\{x_1, x_2, x_3\}$, cast them in matrix form, and solve them. Verify the equation $\mathbf{x} = P\mathbf{x}$ for the resulting solution.

b. Write a for-loop to iterate Eq. (1), say for $k = 0 : 100$, and insert some conditional-if code to break out of the loop as soon as \mathbf{x}_{k+1} is to within 10^{-10} from \mathbf{x}_k , as measured by the Euclidean vector norm, that is, as soon as, $\text{norm}(\mathbf{x}_{k+1} - \mathbf{x}_k) < 10^{-10}$. Print out the resulting maximum number of iterations k and the vector \mathbf{x}_k and compare it with the equilibrium vector \mathbf{x} found in part (a).

c. For the given P , it can be shown that the k th power P^k is given explicitly as the product of matrices:

$$P^k = \frac{1}{48} \begin{bmatrix} 5 & -3 & 3 \\ 8 & 0 & -6 \\ 3 & 3 & 3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & (0.2)^k & 0 \\ 0 & 0 & (0.4)^k \end{bmatrix} \begin{bmatrix} 3 & 3 & 3 \\ -7 & 1 & 9 \\ 4 & -4 & 4 \end{bmatrix}, \quad k = 0, 1, 2, \dots \quad (4)$$

Write a single-line anonymous function $f(k)$ that calculates the matrix in the right-hand side of Eq. (4), and verify, for example, that $f(0) = P^0$, $f(1) = P$, and $f(10) = P^{10}$ and $f(20) = P^{20}$.

d. Using Eq. (4) construct the limiting matrix

$$P_{\text{inf}} = \lim_{k \rightarrow \infty} P^k$$

and discuss its relationship to the equilibrium vector \mathbf{x} .

4. *Mortgage Problem in Matrix Form.* You wish to take a K -year loan in the amount of y_a dollars at a fixed annual percentage rate of R percent to buy a house. The loan payments will be monthly, so that there will be $N = 12K$ payment periods, each at the monthly interest rate of $R/12$ percent, or, in absolute units, $r = R/1200$.

Let $y(k)$ be a Matlab array that denotes the loan balance (i.e., how much you owe the bank) at the beginning of the k th month, and let x denote the fixed amount that you have agreed to pay each month. During the k th month, the bank charges you interest (i.e., it increases the amount you owe) by the amount, $r \cdot y(k)$, and then it subtracts your payment of x dollars for that month. Therefore, the loan balance at the beginning of the next month will be:

$$y(k+1) = y(k) + r \cdot y(k) - x = (1+r)y(k) - x, \quad \text{or,} \\ y(k+1) = ay(k) - x, \quad k = 1, 2, \dots, N \quad (5)$$

where we defined the quantity, $a = 1 + r$. The recursion (5) is initialized at the initial loan amount, $y(1) = y_a$. The amount x is determined by the requirement that the balance be reduced to zero at the end of the N th month, or at the beginning of the $(N+1)$ th month, that is, $y(N+1) = 0$. Summarizing, we have the conditions:

$$y(1) = y_a \\ y(k+1) = ay(k) - x, \quad k = 1, 2, \dots, N \\ y(N+1) = 0 \quad (6)$$

This recursion can be solved in analytical form, resulting into the following expressions for the monthly payment x , and loan balance $y(k)$, in terms of the parameters, y_a, a, N :

$$x = \frac{ry_a a^N}{a^N - 1}, \quad y(k) = y_a \frac{a^N - a^{k-1}}{a^N - 1}, \quad k = 1, 2, \dots, N + 1 \quad (7)$$

It is easily verified that $y(1) = y_a$ and $y(N + 1) = 0$. These are the standard formulas by which the bank calculates your mortgage balance and payment x . However, we wish to solve this problem using a matrix formulation. Eqs. (6) can be regarded as a system of $(N + 2)$ linear equations in the $(N + 2)$ unknowns, $[y(1), y(2), \dots, y(N + 1), x]$. For example, if $N = 5$, we have the $N + 2 = 7$ equations, which, after moving all the terms to the left-hand side, can be rearranged in matrix form:

$$\begin{aligned} y(1) &= y_a \\ y(2) - ay(1) + x &= 0 \\ y(3) - ay(2) + x &= 0 \\ y(4) - ay(3) + x &= 0 \\ y(5) - ay(4) + x &= 0 \\ y(6) - ay(5) + x &= 0 \\ y(6) &= 0 \end{aligned} \Rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -a & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & -a & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & -a & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & -a & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & -a & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} y(1) \\ y(2) \\ y(3) \\ y(4) \\ y(5) \\ y(6) \\ x \end{bmatrix} = \begin{bmatrix} y_a \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (8)$$

or, compactly, $Az = \mathbf{b}$, with solution, $\mathbf{z} = A \setminus \mathbf{b}$, where

$$A = \left[\begin{array}{cccccc|c} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -a & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & -a & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & -a & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & -a & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & -a & 1 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array} \right], \quad \mathbf{z} = \begin{bmatrix} y(1) \\ y(2) \\ y(3) \\ y(4) \\ y(5) \\ y(6) \\ x \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} y_a \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (9)$$

The matrix A and vector \mathbf{b} can be constructed easily by the single Matlab commands,

$$\begin{aligned} A &= [\text{diag}(-a \cdot \text{ones}(5,1), -1) + \text{eye}(6), [0; \text{ones}(5,1)]; \text{zeros}(1,5), 1, 0]; \\ \mathbf{b} &= [1; \text{zeros}(6,1)] * y_a; \end{aligned}$$

The “diag+eye” part constructs the upper-left $(N + 1) \times (N + 1)$ sub-block of A , to which is appended the last column consisting of 0 followed by N ones, and then, the entire bottom row is added.

- Write a one-line anonymous Matlab function of two variables a, N , say, $A = f(a, N)$, that generalizes the above matrix construction to the general case of arbitrary N . It must be defined as a function handle:

$$f = @(a, N) \dots$$

- Suppose you take a 10-year loan of \$100,000 at a fixed annual percentage rate of 6%, so that there will be $N = 10 \times 12 = 120$ payment periods, at the monthly interest rate of 6/12 percent, or, $r = 6/1200 = 0.005$.

Using your function of part (a), construct the loan matrix A and vector \mathbf{b} of the above linear system, and solve it for the monthly payment amount x and the 121-dimensional column vector of balances, $\mathbf{y} = [y(1), y(2), \dots, y(120), y(121)]'$.

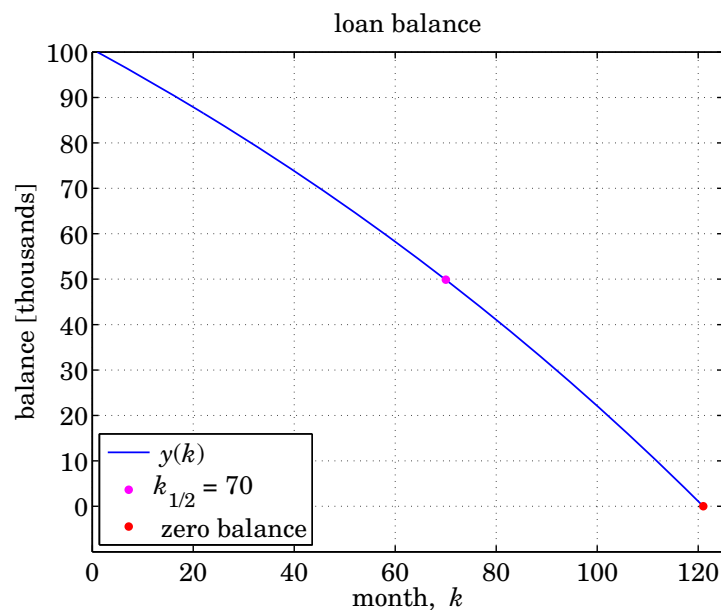
Calculate also the vector of interests, $r \cdot \mathbf{y}$, that the bank charges you at each month, as well as the vector, $x - r \cdot \mathbf{y}$, of the amounts by which your balance is actually being reduced every month (i.e., part of x goes into paying the interest and only the rest goes into reducing the balance.)

- c. Using at most eight **fprintf** commands, generate a table of values that displays the month k , balances $y(k)$, interest, $ry(k)$, and balance reduction, $x - ry(k)$, exactly as shown below, including all the headers and tails, but with all the 121 entries printed.

month k	balance y(k)	interest r*y(k)	reduction x-r*y(k)
1	100000.00	500.00	610.21
2	99389.79	496.95	613.26
3	98776.54	493.88	616.32
4	98160.22	490.80	619.40
...
118	3297.58	16.49	1093.72
119	2203.87	11.02	1099.19
120	1104.68	5.52	1104.68
121	0.00	0.00	1110.21

payment per period:	x =	1110.21
total payments:	N*x =	133224.60
total interest:		33224.60

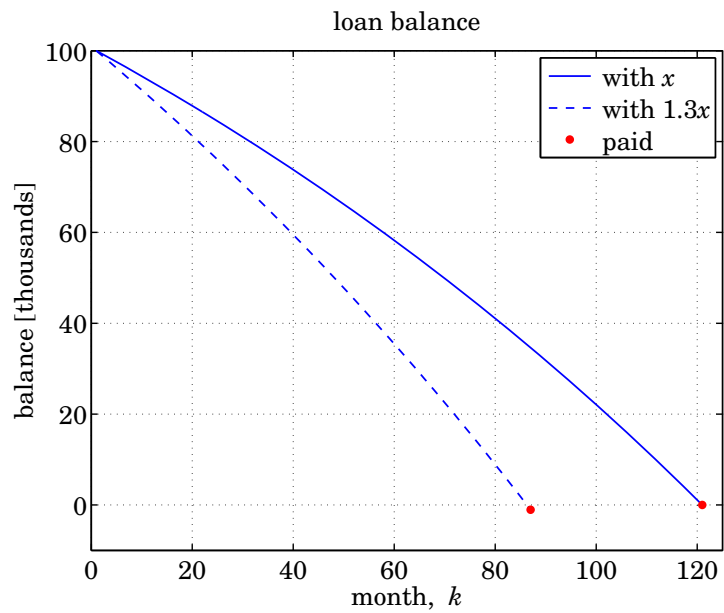
- d. Make a plot of $y(k)$ versus k , for $k = 1, 2, \dots, N + 1$. Add the zero point at the end of your graph, i.e., $y(N + 1) = 0$. Moreover, using the **find** command, determine the earliest month at which you have paid more than half the initial loan, and position it on the graph as shown below.



- e. Recompute the quantities, x, y , using the analytical formulas (7), and compare them to those obtained above by the matrix method. To compare vectors, you may use the function **norm**.
- f. By law you are allowed to pay any additional amount at any time during your mortgage loan in order to pay off the loan faster. For the above example defined in part (b), suppose you decide to pay a little more than the minimum required amount x , say, 30% more each month, that is, $1.3x$. The loan repayment recursion (6) becomes now:

$$\begin{aligned}
 y(1) &= y_a \\
 y(k+1) &= ay(k) - 1.3x, \quad k \geq 1
 \end{aligned}
 \tag{10}$$

Using a *forever* while loop, iterate this recursion and determine the month (and hence the year) at which the balance drops below zero. Make a combined plot that displays the loan balances computed in this part and in part (d), as shown below.



Homework Problems – Week 11
440:127 – Spring 2015 – S. J. Orfanidis

1. End-of-Chapter Problem 13.9.
2. End-of-Chapter Problem 13.10.
3. End-of-Chapter Problem 13.11.
4. When a charged capacitor C discharges through a resistor R , the voltage across the capacitor terminals decays exponentially in time:

$$V(t) = V_0 e^{-at}$$

where V_0 is the initial voltage and $a = 1/(RC)$. Experimental measurements of the voltage at 0.05-second intervals are given in the file **capacitor.dat**. Using these measurements, carry out a least-squares fit to determine the parameters V_0, a . Then, make a plot of the fitted curve versus time, displaying also the measured data, as shown in the graph at the end of this assignment.

5. The file **NYCtemp.dat** contains the monthly mean temperatures of New York City over the five-year period from 1971 to 1975. It is desired to check the validity of the following temperature model that assumes a 12-month periodicity:

$$T(t) = A + B \cdot \cos\left(\frac{2\pi t}{12}\right) + C \cdot \sin\left(\frac{2\pi t}{12}\right)$$

where T is in units of degrees Fahrenheit, and the parameters A, B, C are to be determined using a least-squares fit with trigonometric basis functions.

Let T_i denote the column vector the 60 monthly observations, concatenated for each year, and let t_i denote the column vector of months, $t_i = [0 : 60]'$, with 0 representing Jan. 1971 and 59, Dec. 1975.

- a. Carry out a least-squares fit to determine the parameters A, B, C . Use $\{1, \cos(2\pi t/12), \sin(2\pi t/12)\}$ as the basis functions.
- (a) On the same graph, plot the estimated model $T(t)$ versus t over 600 points in the interval $0 \leq t \leq 59$, and add the actual observations to the graph.
6. The thermal conductivity properties of conductors near absolute zero are important in the design of superconducting systems. The file **copper.dat** contains measurements of the thermal conductivity k of copper in the range $0 \leq T \leq 50$ (in degrees Kelvin). It is desired to use the following model to represent the data:

$$k(T) = \frac{1}{\frac{c_1}{T} + c_2 T + c_3 T^2 + c_4 T^3} \Rightarrow \frac{1}{k(T)} = \frac{c_1}{T} + c_2 T + c_3 T^2 + c_4 T^3$$

- a. Using the basis functions $\{T^{-1}, T, T^2, T^3\}$, perform a least-squares fit to determine the coefficients $\{c_1, c_2, c_3, c_4\}$. Then, plot the model $k(T)$ versus T over the range $0 \leq T \leq 50$ and add the data points to the graph.
- b. Perform the fitting using the built-in function **nlinfit** by defining the fitting function directly from the given model:

$$f(c, T) = \frac{1}{\frac{c_1}{T} + c_2 T + c_3 T^2 + c_4 T^3}, \quad c = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix}$$

Use the solution vector \mathbf{c} of part (a) as the initial vector for **nlinfit**. Determine the parameter vector \mathbf{c} and make a similar plot as in part (a).

7. Practice Exercise 13.1 (p. 492).

8. A projectile launched from zero height with initial velocity v_0 and angle θ_0 follows the trajectory:

$$y = x \tan \theta_0 - \frac{g}{2v_0^2 \cos^2 \theta_0} x^2 \quad (1)$$

where $g = 9.81 \text{ m/s}^2$.

- a. Using $\{x, x^2\}$ as basis functions carry out a least-squares fit to determine the parameters θ_0 (in degrees) and v_0 (in m/sec) given the following eight noisy measurements:

x_i	10.2	14.4	19.8	25.6	29.1	35.3	40.9	43.3
y_i	5.9	7.9	9.1	10.1	10.4	10.7	9.2	8.3

Using the fitted values, make a plot of y vs. x from Eq. (1) over the interval $0 \leq x \leq 60$ and add the data points $\{x_i, y_i\}$ to the graph.

- b. Repeat part (a) but now use the **polyfit** function. Explain any slight differences in the results.

[For your reference, the exact values were $\theta_0 = 35^\circ$ and $v_0 = 25 \text{ m/sec}$.]

9. The atmosphere gets thinner with height. The following measurements of the air density ρ in kg/m^3 versus height h in km are given:

h_i	0	3	6	9	12	15	18	21	24	27	30	33
ρ_i	1.2	0.92	0.66	0.47	0.31	0.19	0.12	0.075	0.046	0.029	0.018	0.011

- a. Make a preliminary plot of ρ_i versus h_i to get an idea of how the density depends on height. You will notice that very likely an exponential model of the following form is appropriate:

$$\rho = \rho_0 e^{-h/h_0} \quad (2)$$

Using a least-squares fit, determine estimates of the parameters ρ_0 and h_0 . Make a plot of the estimated model of Eq. (2) over the range $0 \leq h \leq 35$ and add the data to the graph. Also, make a plot of $\log(\rho)$ versus h and add the data to it.

- b. Repeat part (a) using the higher order model:

$$\rho = \exp(c_1 h^2 + c_2 h + c_3) \quad (3)$$

- c. Determine estimates of the air density at heights of 5 and 10 km using the following three methods: (i) using your model of Eq. (3), (ii) using the function **spline**, and (iii) using **pchip**.

10. The viscosity of water is modeled as a function of temperature by Andrade's equation:

$$\nu = \exp\left(A + \frac{B}{T}\right) \quad (4)$$

where T is the temperature in absolute units, i.e., related to degrees Celsius by $T = 273.15 + T_C$, and A, B are constants. Eq. (4) can be written in the following form:

$$\ln(\nu) = A + \frac{B}{T} \quad (5)$$

The following data are given,[†] where T_C is in degrees Celcius and ν in $\text{milli-N}\cdot\text{s/m}^2$, and for convenience, they can be loaded from the file, `water_visc.dat`:

[†]Table B.2, Appendix B, in B. R. Munson, et al., *Fundamentals of Fluid Mechanics*, 7/e, Wiley, New York, 2013.

T_{Ci}	v_i
0	1.7870
5	1.5190
10	1.3070
20	1.0020
30	0.7975
40	0.6529
50	0.5468
60	0.4665
70	0.4042
80	0.3547
90	0.3147
100	0.2818

- a. Using Eq. (5) and the **polyfit** function, perform a least-squares fit to determine the parameters A, B . Then, use these parameters in Eq. (4) and plot v versus T_C in the interval $0 \leq T_C \leq 100$ °C and add the data points to the graph.
- b. Perform the fitting using the built-in function **nlinfit** or **lsqcurvefit** by defining the fitting function directly from Eq. (4):

$$f(c, T) = \exp\left(c_1 + \frac{c_2}{T}\right), \quad c = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$$

Determine the parameter vector c and make a similar plot as in part (a).

- c. Collect the results of the two methods and with at most three **fprintf** commands, print a table exactly as shown below:

A	B	method
-6.3462	1871.0684	polyfit
-6.8480	2020.9920	nlinfit

- d. As a generalization of Andrade's equation, consider the model,

$$v = \exp\left(A + \frac{B}{T} + \frac{C^2}{T^2}\right) \quad (6)$$

Using the **polyfit** function, perform a least-squares fit to determine the parameters A, B, C . Then, use these parameters in Eq. (6) and plot v versus T_C in the interval $0 \leq T_C \leq 100$ °C and add the data points to the graph. Finally, make a table like the one below.

A	B	C	method
-1.1615	-1427.7335	720.6126	polyfit-2

11. *Sprinting Models.* Simple mathematical models exist for fitting the track data of 100-meter sprinters. The following two-parameter and three-parameter models, due to Keller and Tibshirani (see references at end and papers on sakai), fit very well the track data of some very famous sprinters, such as Usain Bolt (currently considered to be the fastest man in the world), Carl Lewis, and Ben Johnson. Let $v(t)$ and $x(t)$ denote the speed and distance traveled at time t ,

$v(t) = \alpha \left(1 - e^{-\gamma(t-t_0)}\right)$ $x(t) = \alpha(t - t_0) - \frac{\alpha}{\gamma} \left(1 - e^{-\gamma(t-t_0)}\right)$	(Keller)	(7)
--	----------	-----

$$\boxed{\begin{aligned} v(t) &= \alpha \left(1 - e^{-\gamma(t-t_0)}\right) - \beta t \\ x(t) &= \alpha(t-t_0) - \frac{1}{2}\beta(t-t_0)^2 - \frac{\alpha}{\gamma} \left(1 - e^{-\gamma(t-t_0)}\right) \end{aligned}} \quad \text{(Tibshirani)} \quad (8)$$

where $t \geq t_0$, and α, β, γ are parameters to be fitted, and t_0 is the reaction time (which legally must be greater than 0.1 sec to qualify.) The Keller model corresponds to setting $\beta = 0$ in the Tibshirani one. Physically, the two models arise from the following equations of motion, both of which assume a frictional force proportional to $-\gamma v$, and a constant accelerating force in the Keller model, or a linearly decreasing one in the Tibshirani one (i.e., the sprinter can't maintain a constant force for the entire run),

$$\frac{dv}{dt} = -\gamma v + F \quad \text{(Keller)}$$

$$\frac{dv}{dt} = -\gamma v + (F - ct) \quad \text{(Tibshirani)}$$

where F, c are the constants, $F = \alpha\gamma - \beta$, $c = \gamma\beta$. In sprinting events, the elapsed time is recorded at 10-meter intervals, for example, for Usain Bolt at Beijing 2008, we have the observed data:

x_i (meters)	0	10	20	30	40	50	60	70	80	90	100
t_i (sec)	0.165	1.85	2.87	3.78	4.65	5.50	6.32	7.14	7.96	8.79	9.69

One of the α, β, γ parameters can be fixed in terms of the other ones by requiring that the model match one of the data points exactly, for example, at the ending distance of $x_e = 100$ meters, and corresponding ending time, e.g., $t_e = 9.69$ sec. For the Tibshirani model, we have:

$$x_e = \alpha(t_e - t_0) - \frac{1}{2}\beta(t_e - t_0)^2 - \frac{\alpha}{\gamma} \left(1 - e^{-\gamma(t_e - t_0)}\right) \Rightarrow \alpha = \frac{x_e + \frac{1}{2}\beta(t_e - t_0)^2}{t_e - t_0 - \frac{1}{\gamma} \left(1 - e^{-\gamma(t_e - t_0)}\right)} \quad (9)$$

Define the parameter vector in terms of the remaining parameters, β, γ :

$$c = \begin{bmatrix} \beta \\ \gamma \end{bmatrix} \quad \text{(Tibshirani)}, \quad c = \gamma \quad \text{(Keller)}$$

so that $c(1) = \beta$, $c(2) = \gamma$, or, in the Keller case, $c(1) = \gamma$.

- a. Define a function, say $A(c)$, of the parameter vector c defined above that implements Eq. (9). It must be defined as an anonymous MATLAB function, i.e.,

$$A = @(c) \dots$$

Then, using this function, define two anonymous MATLAB functions, $x(c, t)$, $v(c, t)$, of the independent variables c, t , with t being vectorized, implementing Eqs. (7) or (8), i.e.,

$$x = @(c, t) \dots$$

$$v = @(c, t) \dots$$

- b. The data file, **sprint.dat**, contains the 100-meter track data for Bolt (two events), Lewis, and Johnson. The first column holds the distances x_i at the 10-meter intervals, and the other columns contain the corresponding recorded times t_i , the top rows being the reaction times t_0 . The arrays x_i, t_i both have length 11.

For each of the four events, starting out with the Tibshirani model and using the function **nlinfit**, perform a least-squares fit with the fitting function $x(c, t)$ defined in part (a) to determine the parameter vector c . You may use the following initial vector $c_0 = [0.01; 1]$.

Once the parameter vector c is known, calculate the predicted or fitted values of t_i that correspond to the distances x_i . This can be done by using the function **fzero** to solve the equation $x(c, t) = x_i$ for t . You may use the observed times t_i as the initial search points for **fzero**. For example, to find the fitted time to reach 50 meters, i.e., with $x_i(6) = 50$, use the code:

```
tf(6) = fzero(@(t) f(c,t)-xi(6), ti(6));
```

Collect together the fitted results for all four events and make a formatted table exactly as shown below using at most seven **fprintf** commands.

	U. Bolt		U. Bolt		C. Lewis		B. Johnson	
x (m)	t (s)	fit (s)	t (s)	fit (s)	t (s)	fit (s)	t (s)	fit (s)
0	0.165	0.165	0.142	0.142	0.193	0.193	0.129	0.129
10	1.85	1.91	1.89	1.91	1.94	1.96	1.84	1.87
20	2.87	2.88	2.88	2.89	2.96	2.97	2.86	2.87
30	3.78	3.77	3.78	3.78	3.91	3.89	3.80	3.78
40	4.65	4.63	4.64	4.63	4.78	4.77	4.67	4.66
50	5.50	5.47	5.47	5.46	5.64	5.64	5.53	5.52
60	6.32	6.31	6.29	6.29	6.50	6.51	6.38	6.38
70	7.14	7.15	7.10	7.11	7.36	7.36	7.23	7.24
80	7.96	7.99	7.92	7.93	8.22	8.22	8.10	8.11
90	8.79	8.84	8.75	8.76	9.07	9.08	8.96	8.97
100	9.69	9.69	9.58	9.58	9.93	9.93	9.83	9.83

Moreover using at most three **fprintf** commands, print the values of the fitted parameters for all four events, as shown below,

a	b	c	sprinter
12.4944	0.0810	0.8144	U. Bolt
12.4024	0.0284	0.7851	U. Bolt
11.5727	-0.0153	0.8628	C. Lewis
11.8391	0.0268	0.8711	B. Johnson

For each event, plot the fitted model $x(c, t)$ versus t over the range, $t_0 \leq t \leq 10$ sec, and add the observed times t_i , as well as the fitted times t_f on the graph (see example graphs at end).

Moreover, plot the velocity function $v(c, t)$ versus t and add the observed average velocities over the 10-meter intervals, which may be computed using the **diff** function by

```
vi = diff(xi)./diff(ti)
```

c. Repeat part (b) for the Keller model.

Both models fit the data very well. Negative values of β , as in the Lewis case, do not make sense (i.e., it would mean an increasing accelerating force towards the end of the run). Therefore, it appears that the Keller model of better suited for the Lewis data.

Sprinting References

1. J. B. Keller, "Theory of Competitive Running," *Phys. Today*, **26**, no.9, p.43 (1973), see also, "Optimal Velocity in a Race," *Am. Math. Monthly*, **81**, 474 (1974).
2. R. Tibshirani, "Who is the Fastest Man in the World?," *Amer. Statistician*, **51**, 106 (1997).
3. G. Wagner, "The 100-Meter Dash: Theory and Experiment," *Phys. Teacher*, **36**, 144 (1998).

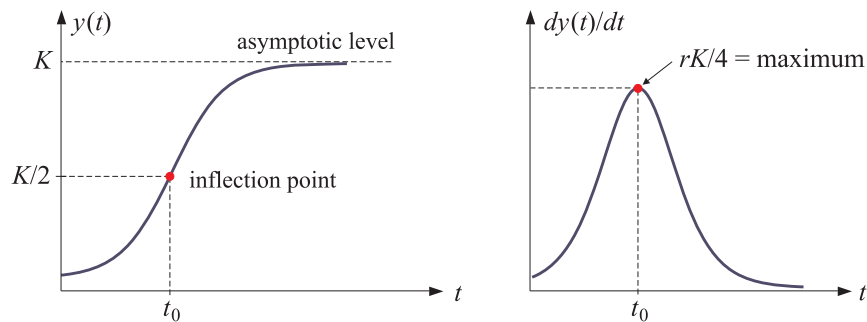
4. O. Helene and M. T. Yamashita, "The Force, Power, and Energy of the 100 Meter Sprint," *Am. J. Phys.*, **78**, 307 (2010).
5. <http://myweb.lmu.edu/jmureika/track/>, J. R. Mureika. Contains additional data.

12. *Growth Models.* The logistic function, originally proposed by P. Verhulst in 1838, has been very successful in describing the growth of populations of humans, plants and animals, algae, bacteria, and the spread of diseases, such as SARS and Measles, as well forecasting the growth and decline of technology products, ideas, and innovations, viewing the economy and marketplace as an "ecosystem". It attempts to model the increase of populations under the constraint of limited natural resources.

The logistic function, as well as some variants that often work better, are listed in the table below, together with the differential equations from which they arise. The typical shape of these growth curves is illustrated in the picture below (for the logistic case).

Model	Function	Differential Equation
Logistic	$y(t) = \frac{K}{1 + e^{-r(t-t_0)}}$	$\frac{dy}{dt} = ry \left[1 - \frac{y}{K} \right]$
Richards	$y(t) = \frac{K}{[1 + \alpha e^{-r(t-t_0)}]^{\frac{1}{\alpha}}}$	$\frac{dy}{dt} = \frac{r}{\alpha} y \left[1 - \left(\frac{y}{K} \right)^\alpha \right]$
Pearl-Reed	$y(t) = \frac{K}{1 + A e^{-r(t)}}$ $r(t) = r_1 t + r_2 t^2 + r_3 t^3$	$\frac{dy}{dt} = \dot{r}(t) y \left[1 - \frac{y}{K} \right]$ $\dot{r}(t) = \frac{dr}{dt} = r_1 + 2r_2 t + 3r_3 t^2$
Gompertz	$y(t) = K \exp \left[-e^{-r(t-t_0)} \right]$	$\frac{dy}{dt} = -r y \ln \left(\frac{y}{K} \right)$
Bass	$y(t) = pK \frac{1 - e^{-(p+q)t}}{p + q e^{-(p+q)t}}$	$\frac{dy}{dt} = [pK + qy] \left[1 - \frac{y}{K} \right]$

where the parameters, $K, r, t_0, \alpha, A, r_1, r_2, r_3, p, q$, are constants to be determined by data fitting. The main feature of the differential equations is that they differ from a pure exponential growth, $dy/dt = ry$, by having an effective growth rate that becomes smaller and smaller as the population increases and it encounters limited resources, e.g., the effective growth rate of the logistic case is $r_{\text{eff}} = r(1 - y/K)$, which becomes smaller as y increases towards its limiting value of K .



The inflection point at $t = t_0$ corresponds the maximum value of the derivative, i.e., the maximum rate of growth. In particular, we have for the logistic, Richards, and Gompertz cases,

Logistic	Richards	Gompertz	
$y(t_0) = \frac{K}{2}$	$y(t_0) = \frac{K}{[1 + \alpha]^{\frac{1}{\alpha}}}$	$y(t_0) = \frac{K}{\exp(1)}$	(11)
$\frac{dy(t_0)}{dt} = \frac{rK}{4}$	$\frac{dy(t_0)}{dt} = \frac{rK}{[1 + \alpha]^{\frac{1}{\alpha} + 1}}$	$\frac{dy(t_0)}{dt} = \frac{rK}{\exp(1)}$	

We note that the logistic and Gompertz curves are special cases of the Richards curve for the values $\alpha = 1$ and $\alpha = 0$, respectively. The Bass model, which is widely used in Marketing Engineering, will be considered in greater detail in the week-12 homework set.

The basic strategy for fitting these growth curves to data is to define the growth function as an anonymous MATLAB function of the desired parameters and time, and then apply the `nlinfit` function. For example, with proper identification of the parameters c , these MATLAB functions are,

```
f = @(c,t) c(1)./(1 + exp(-c(2)*(t-c(3)))); % Logistic
f = @(c,t) c(1)./(1 + c(4)*exp(-c(2)*(t-c(3))))^(1/c(4)); % Richards
f = @(c,t) c(1)./(1 + c(2) * exp(-(c(3)*t + c(4)*t.^2 + c(5)*t.^3))); % Pearl-Reed
f = @(c,t) c(1)*exp(-exp(-c(2)*(t-c(3)))); % Gompertz
```

Given a set of observed data points $(t_i, y_i), i = 1, 2, \dots, N$, the estimated parameters are obtained by

```
c = nlinfit(ti, yi, f, c0);
```

where the initial parameter vector can be estimated as follows:

```
[diff0, i0] = max(diff(yi)./diff(ti)); % maximum derivative
t0 = ti(i0); % inflection point
K0 = max(yi); % limiting level
r0 = 4*diff0/K0; % rate, from Eq.(11)
c0 = [K0; r0; t0]; % parameter vector (Logistic, Gompertz cases)
```

and in addition, set $a = 1$, in the Richards case, and, $A = \exp(r_0 t_0), r_2 = r_3 = 0$, in the Pearl-Reed case. This fitting strategy will be applied to the following data files attached to this homework set, with data sources and references therein:

- (a) **uspop.dat** US population and projections to 2060
- (b) **worldpop.dat** UN projections of the world population to 2100
- (c) **yeast.dat** growth of yeast cells
- (d) **algae.dat** growth of algae
- (e) **measles.dat** measles epidemic
- (f) **sars.dat** SARS epidemic
- (g) **squash.dat** growth of squash plant
- (h) **worldgdp.dat** World GDP
- (i) **usgdp.dat** United States GDP

- a. Load the US population data from the file **uspop.dat**, as well as the US Census predictions for 2015-2060 from the file **uspred.dat**. Fit the logistic model to the data in **uspop.dat**, using all data with a starting date of 1790. Plot both the data, the fitted logistic curve, and the predictions till 2060.

Next, fit the logistic model to the data in **uspop.dat**, using all data with a starting date of 1850, and repeat the above plots.

Finally, fit the Gompertz model to the data in **uspop.dat**, using all data with a starting date of 1790, and generate a similar plot.

- b. Load the World population data from the file **worldpop.dat**, and fit a logistic curve. Plot the data and model curve till the year 2100. The nearly perfect fit of the logistic curve makes it clear that the UN future projections are based on this model.
- c. Fit a logistic model to the yeast data in **yeast.dat** and plot data and model on the same graph.
- d. Load the algae biomass data from the file **algae.dat**. First, fit a logistic model to the data and plot the data and model on the same graph. Then, repeat the plot by fitting the data to a Pearl-Reed model.
- e. Fit the measles epidemic data, **measles.dat**, to both a logistic and a Richards model, and in each case plot the data and model on the same graph.
- f. Fit the SARS epidemic data, **sars.dat**, to all four models (Logistic, Richards, Pearl-Reed, Gompertz), and in each case plot the data and model on the same graph. See example graphs at end.
- g. Fit the squash growth data, **squash.dat**, to all four models (Logistic, Richards, Pearl-Reed, Gompertz), and in each case plot the data and model on the same graph. See example graphs at end.
- h. Fit the World GDP data, **worldgdp.dat**, to both a logistic and a Richards curve. Plot the growth curves to the year 2100 and add the data to the graphs. Note how different the future predictions are from the two methods—which one is right?
- i. Repeat Part (h) for the US GDP data, **usgdp.dat**.

Growth Model References

1. F. Cavallini, "Fitting a Logistic Curve to Data," *Coll. Math. J.*, **24**, 247 (1993).
 2. E. Rozema, "Epidemic Models for SARS and Measles," *Coll. Math. J.*, **38**, 246 (2007).
 3. A. Tsoularis and J. Wallace, "Analysis of Logistic Growth Models," *Math. Biosci.*, **179**, 21 (2002).
 4. R. Pearl, "The Growth of Populations," *Quart. Rev. Biol.*, **2**, 532 (1927). See also, R. Pearl and L. J. Reed, "Skew-Growth Curves," *Proc. Natl. Acad. Sci.* **11**, 16 (1925).
 5. F. J. Richards, "A Flexible Growth Function for Empirical Use," *J. Exp. Botany*, **10**, 290 (1959).
 6. P. Pflaumer, "Forecasting the U.S. Population with the Gompertz Growth Curve," *Proc. Joint Statistical Meetings*, Social Statistics Section, San Diego 2012, available online from: <http://home.arcor.de/peteroskar/Homepage/ASA2012.pdf>
 7. G. P. Boretos, "The future of the global economy," *Technol. Forecasting & Soc. Change*, **76**, 316 (2009).
13. *Life Tables*. The CDC maintains and regularly updates vital statistics for the US population. The file, **lifetbl.dat**, contains the most recent mortality and life tables as of 2009. The first column represents the age of an individual in the range, $0 \leq t \leq 100$ years. The second column, labeled $q(t)$, represents the probability of dying between years t and $t + 1$, and is a measure of the mortality of an individual. The quantity, $p(t) = 1 - q(t)$, is the probability of survival between years t and $t + 1$. The Gompertz law of mortality, proposed in 1825, fits the survival probability (and hence $q(t)$) to a Gompertz function of the form:

$$\begin{aligned}
 p(t) &= \exp[-A \exp(r(t - t_0))] \\
 q(t) &= 1 - p(t) = 1 - \exp[-A \exp(r(t - t_0))]
 \end{aligned}
 \tag{12}$$

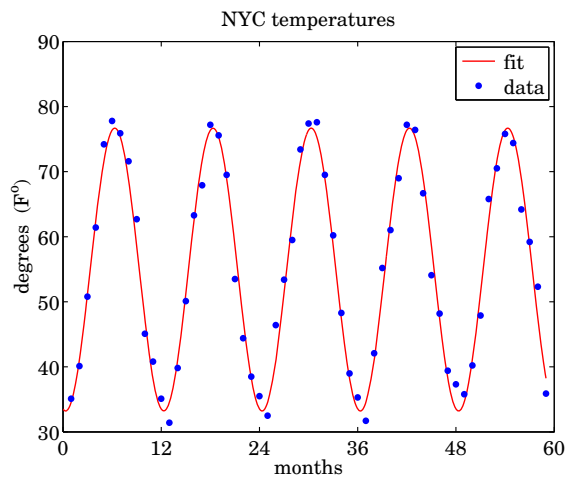
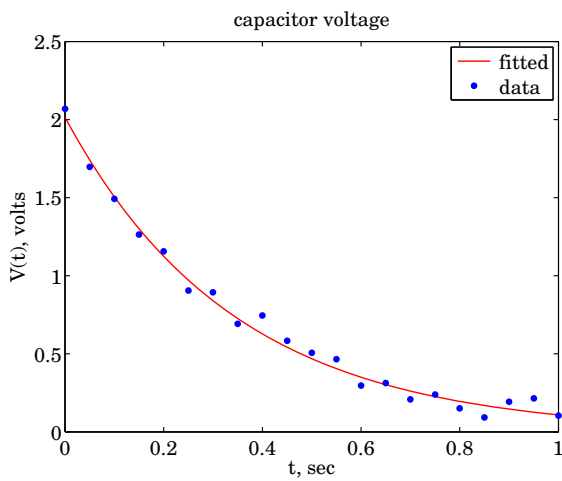
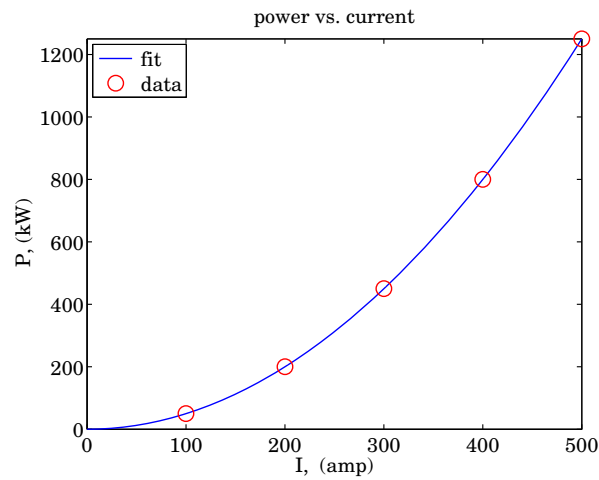
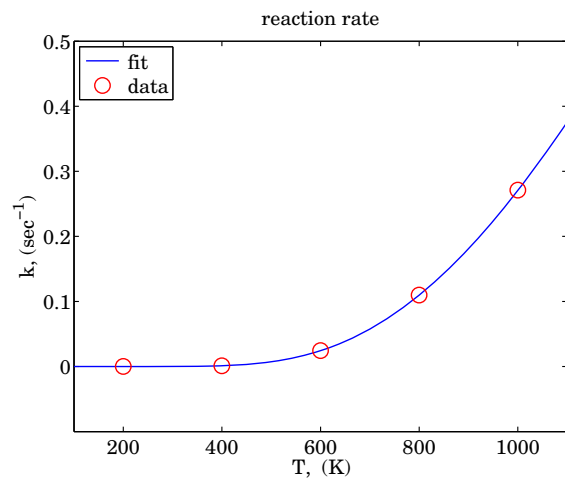
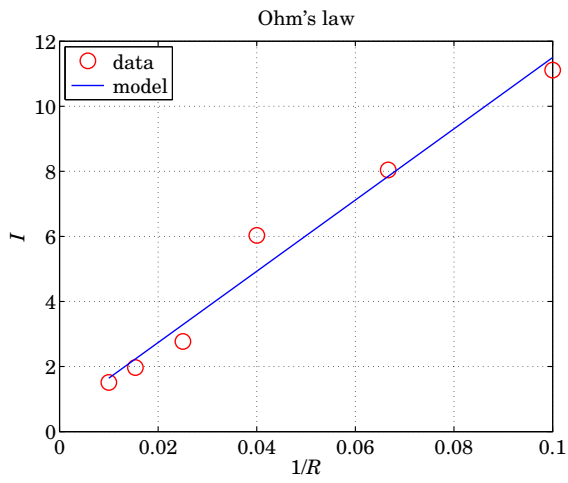
where A, r, t_0 are parameters to be fitted. We note that t_0 is not an independent parameter since it combines with A as a common factor Ae^{-rt_0} . Therefore, t_0 can be fixed to some convenient value, such as, $t_0 = 25$ years. Note the difference in the sign of r in Eqs. (10) and (12), the former representing growth, and the latter, rapid decay. In MATLAB, the function $q(t)$ can be defined as a function of the parameters A, r, t :

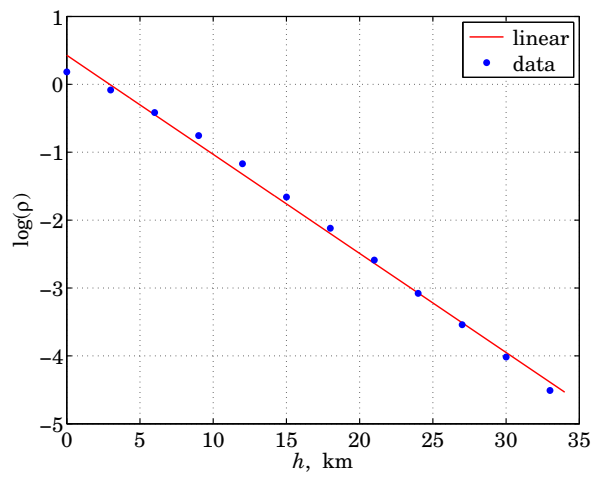
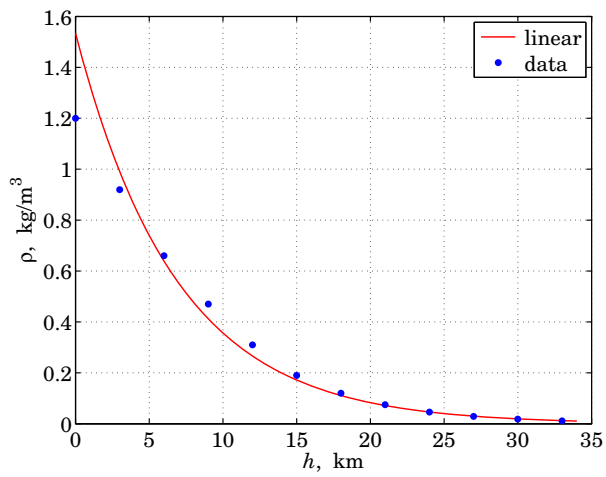
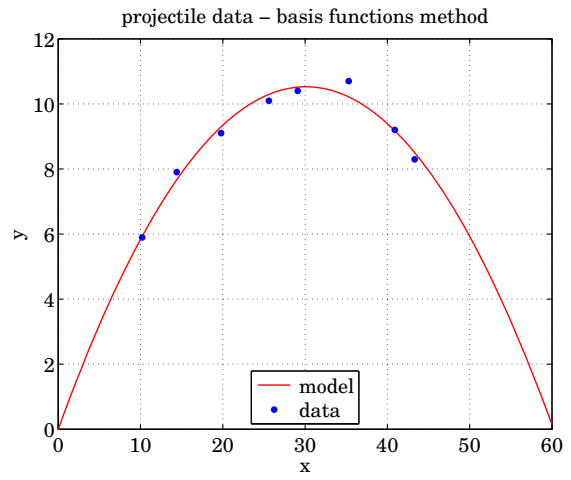
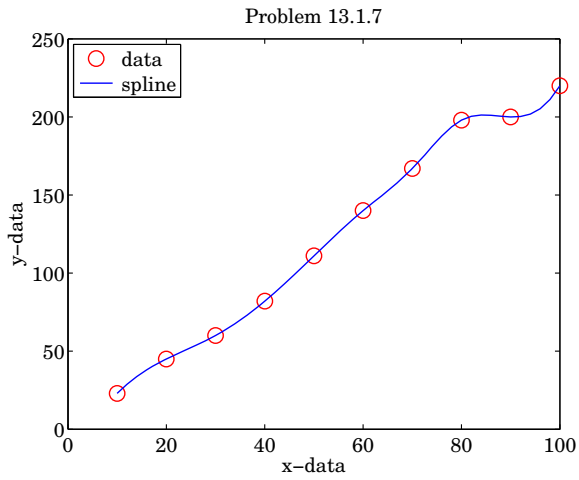
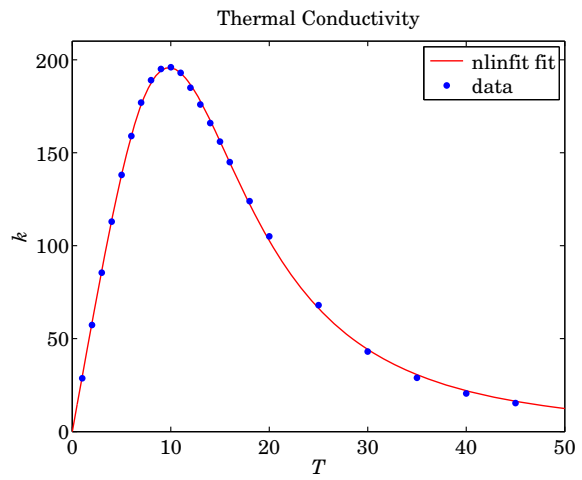
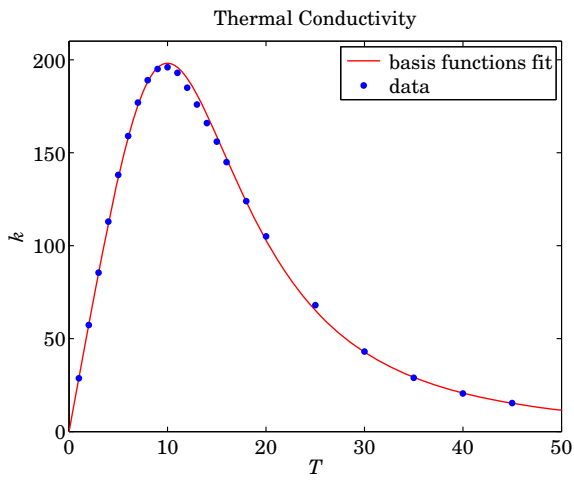
```

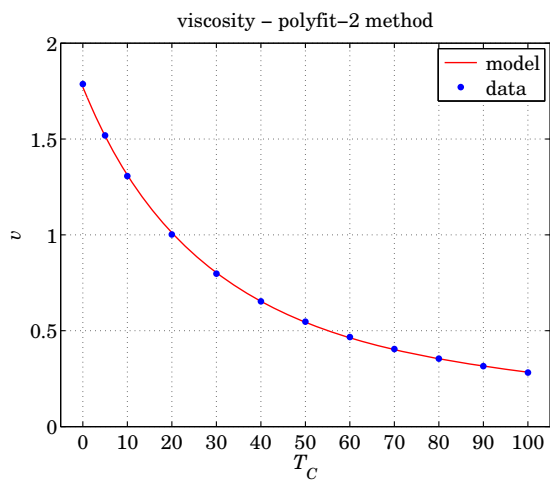
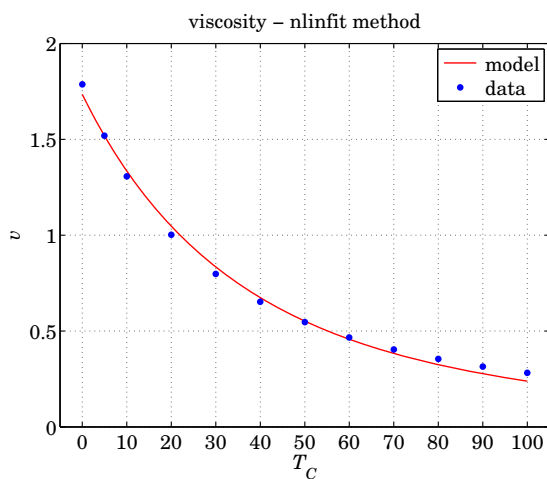
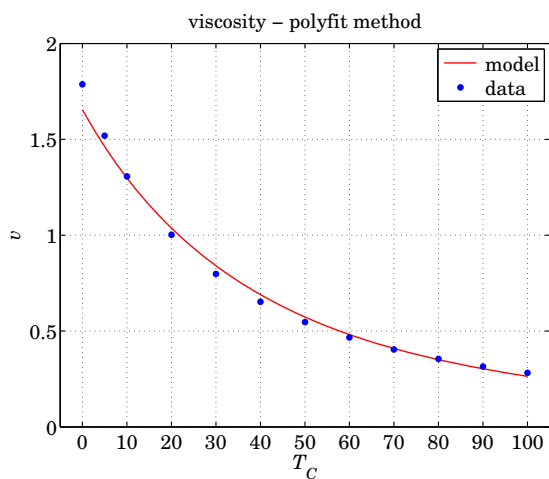
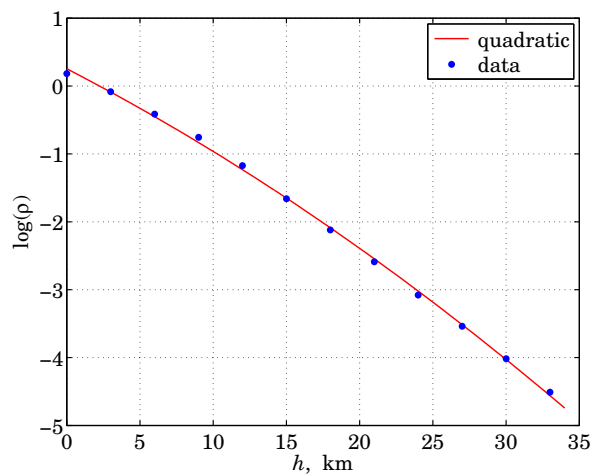
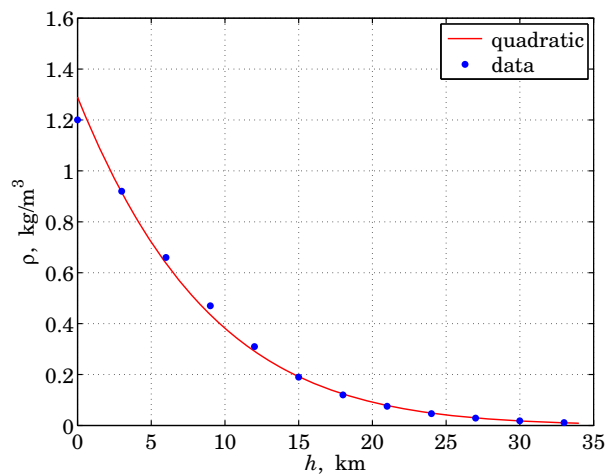
t0 = 25;
Q = @(c,t) 1 - exp(-c(1)*exp(c(2)*(t-t0)));

```

- a. Load the data from **lifetbl.dat**, but exclude the last data point at $t = 100$ since the probability of dying at that age is artificially taken to be 100%.
Using **nlinfit**, fit the function $Q(c, t)$ to the probability data $q(t)$ and determine the parameters A, r .
Make a plot of the model versus time and add the data to the graph.
- b. In plotting the above on a **semilogy** axis, one notes that the fitting function follows essentially a straight line with a slope of about 0.0430, that is, in absolute scales, $10^{0.043t}$. But we note that approximately, $10^{0.043t} = 2^{t/7}$, which implies that the probability of death doubles every seven years. To check this, let Q_0 be the value of the fitted function at $t = 0$, that is, $Q_0 = Q(c, 0)$. Plot the normalized quantity $\log_2(Q(c, t)/Q_0)$ versus t and add the data to the graph, i.e., $\log_2(q(t)/Q_0)$, as well as the straight line, $y = t/7$. See example graph at end.

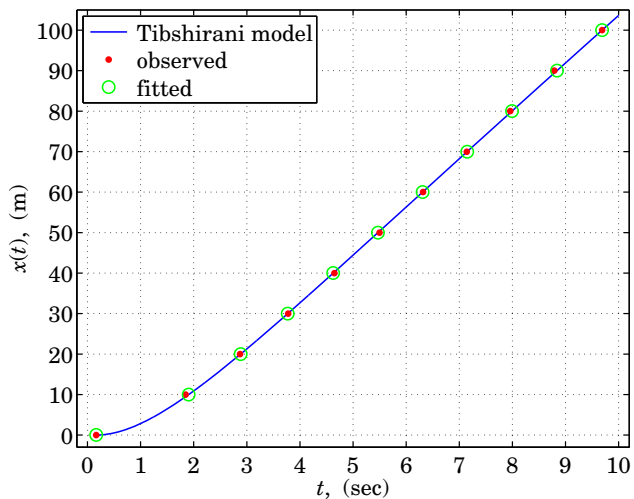




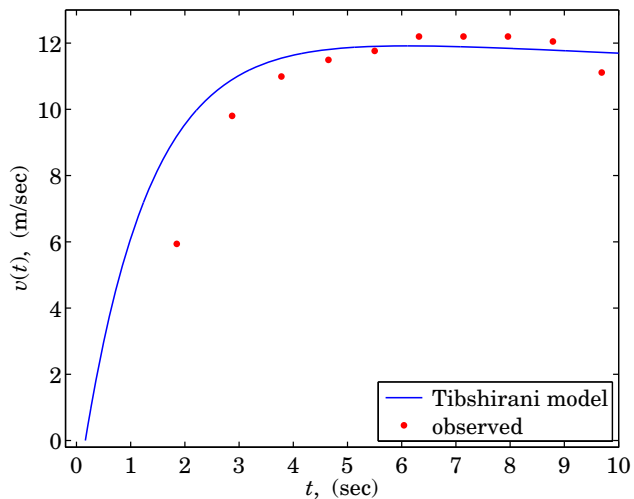


Sprinting Models

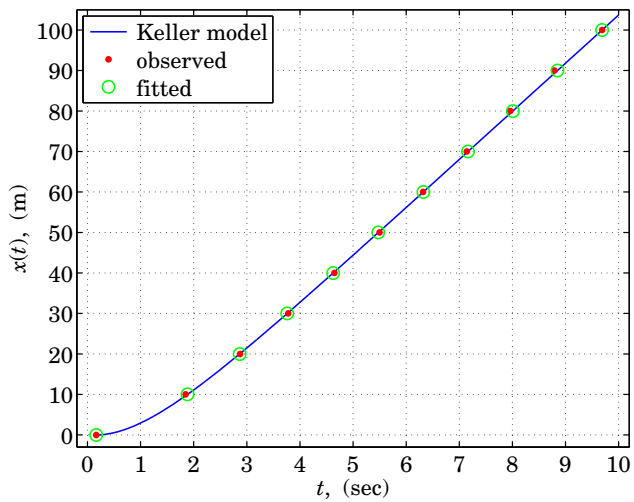
U. Bolt / Beijing 2008



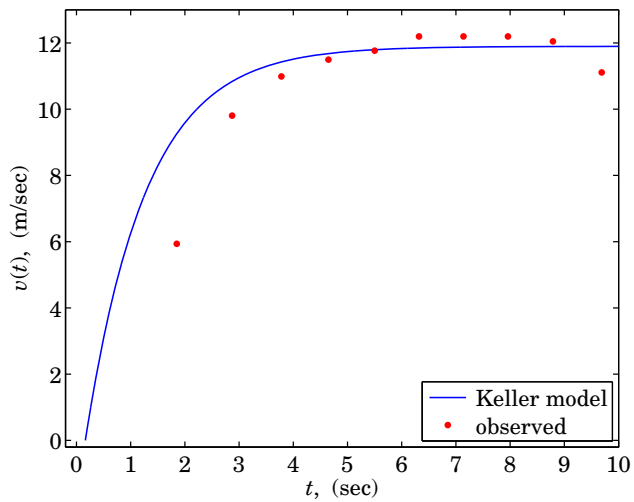
U. Bolt / Beijing 2008

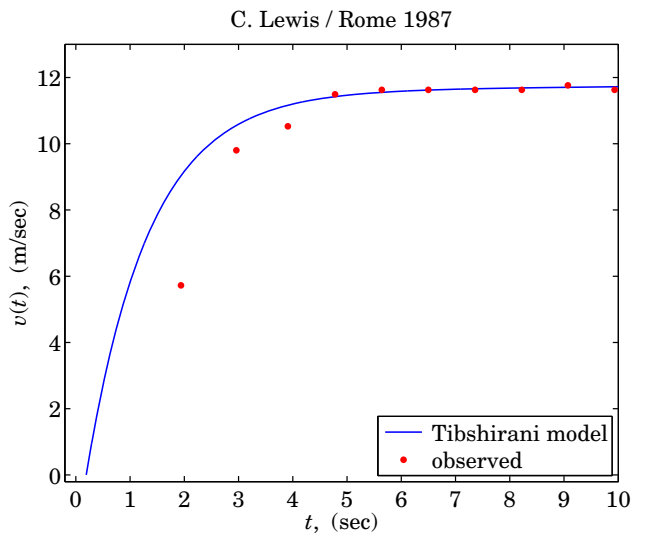
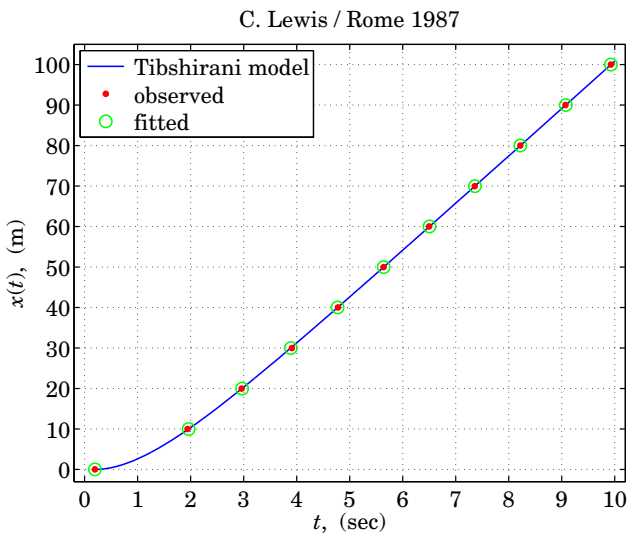
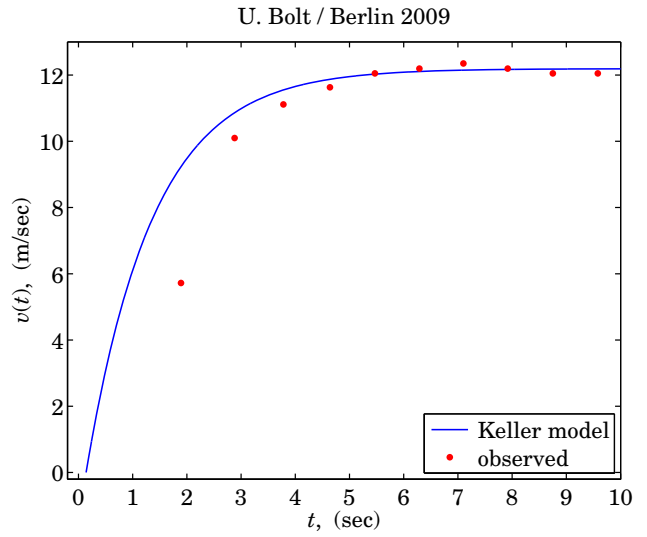
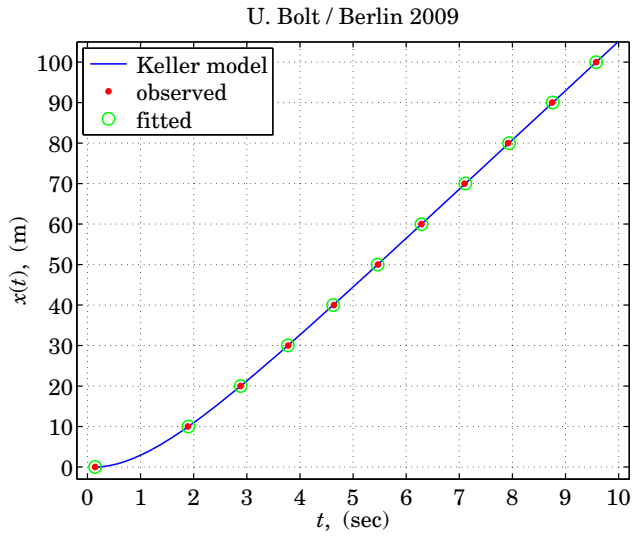
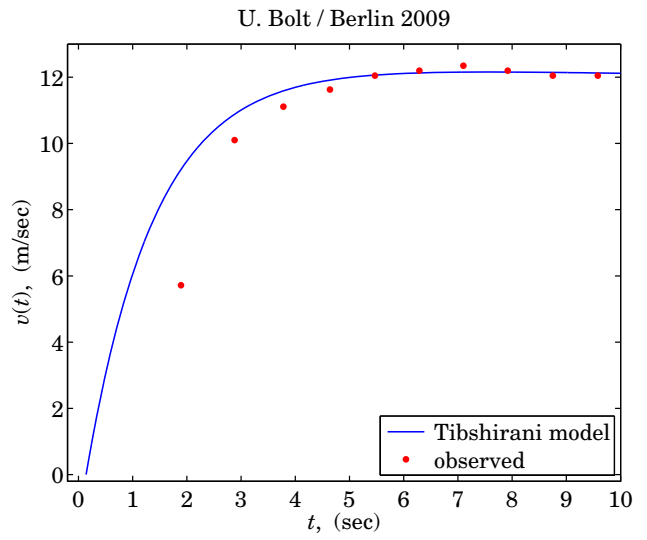
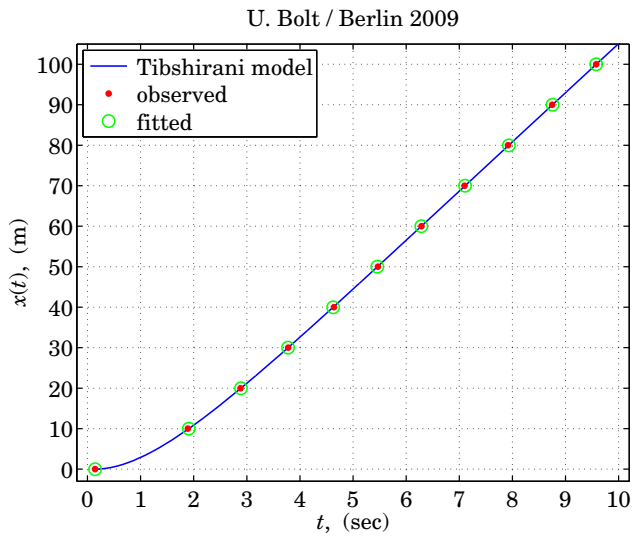


U. Bolt / Beijing 2008

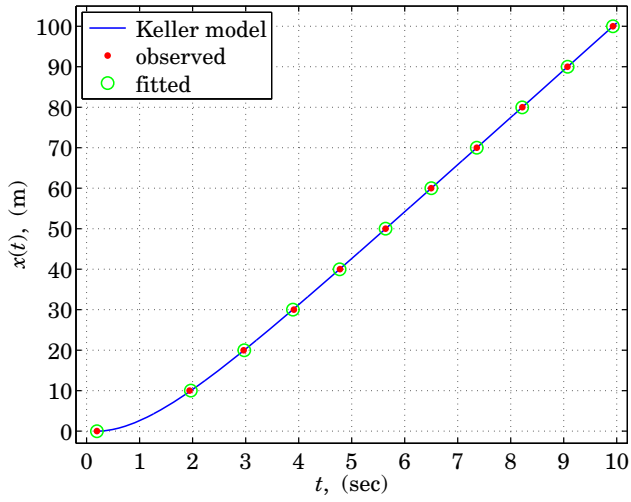


U. Bolt / Beijing 2008

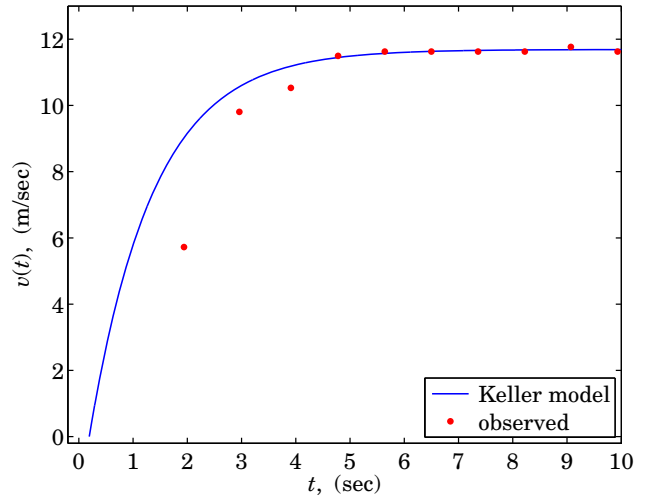




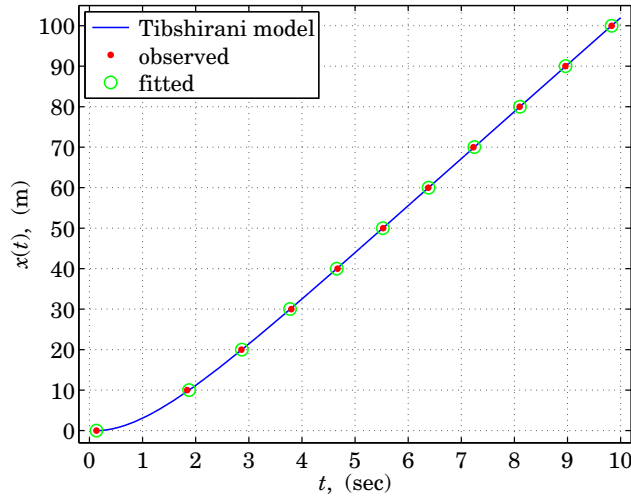
C. Lewis / Rome 1987



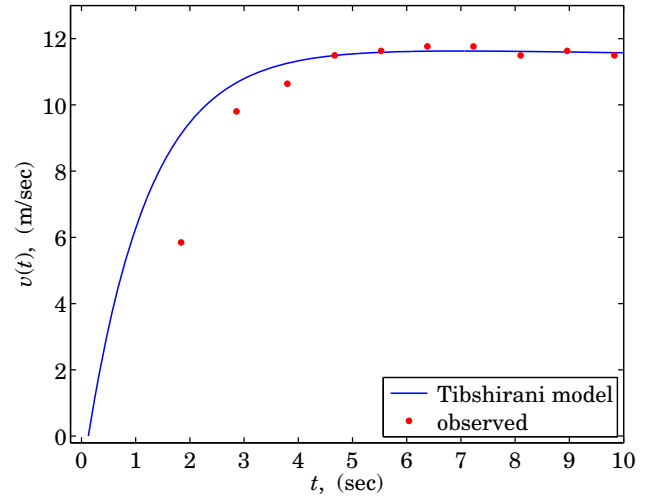
C. Lewis / Rome 1987



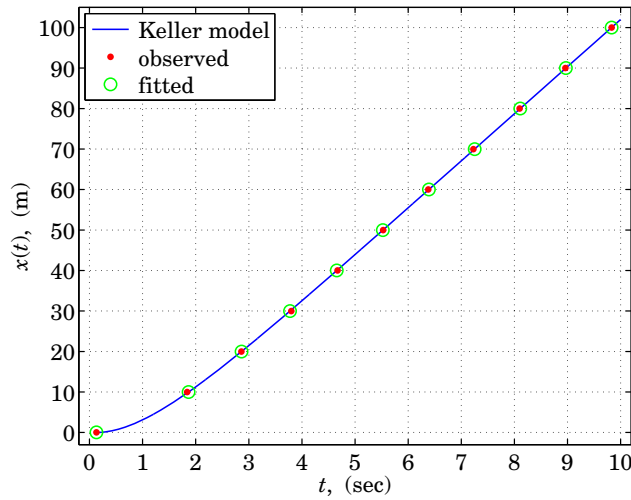
B. Johnson / Rome 1987



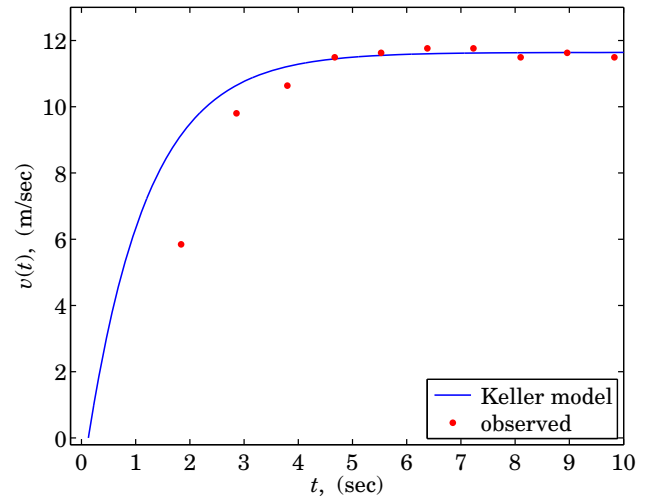
B. Johnson / Rome 1987



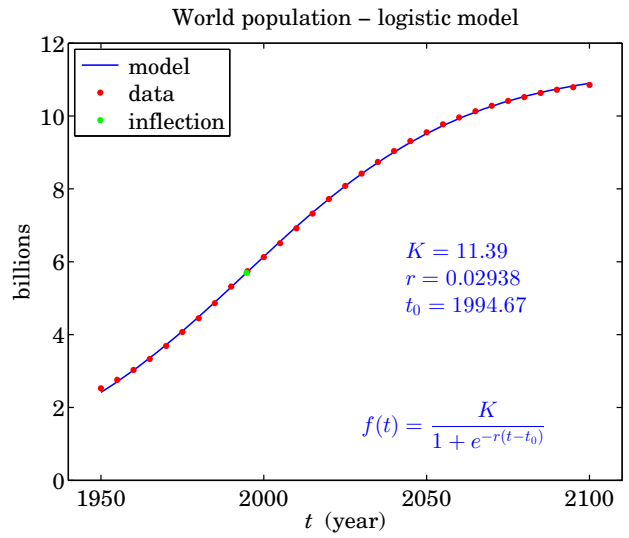
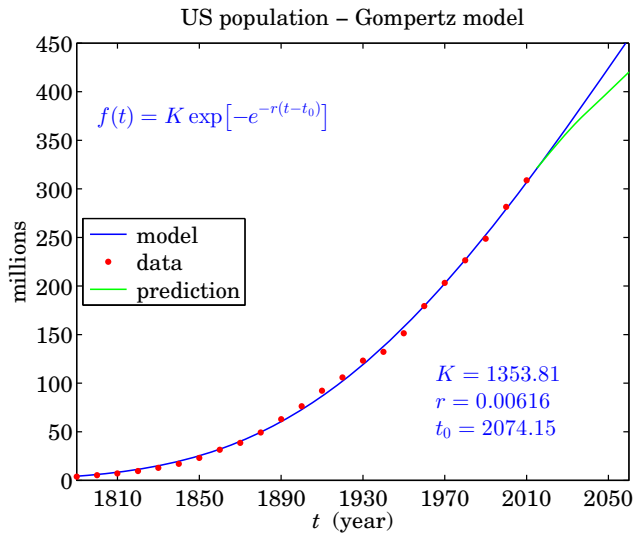
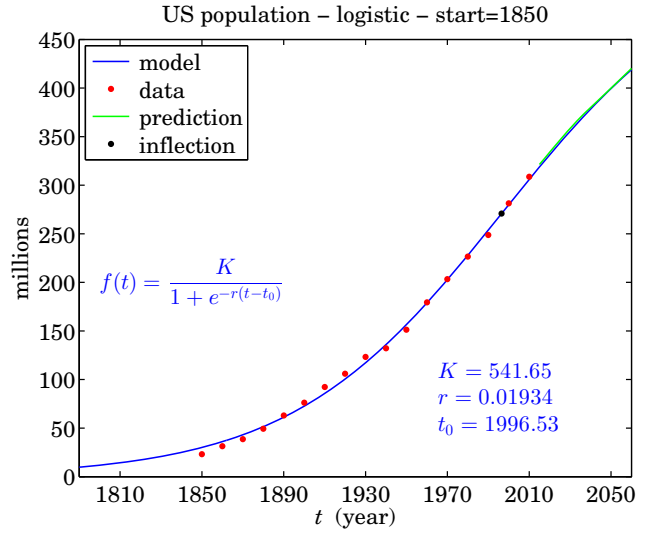
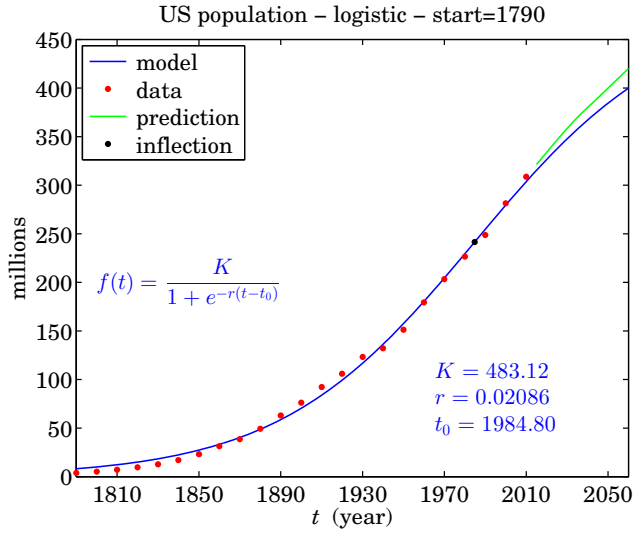
B. Johnson / Rome 1987



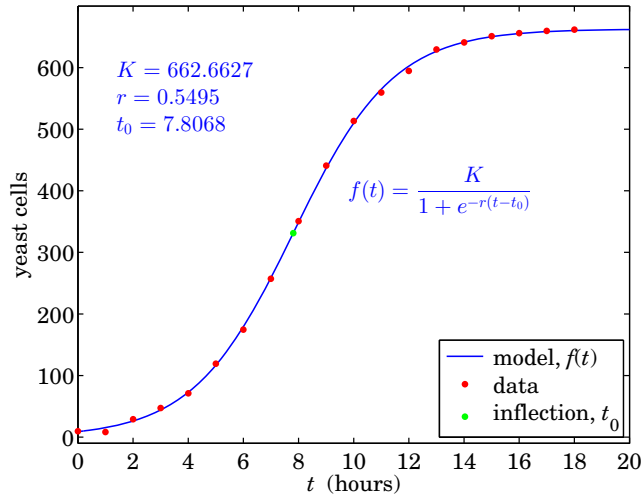
B. Johnson / Rome 1987



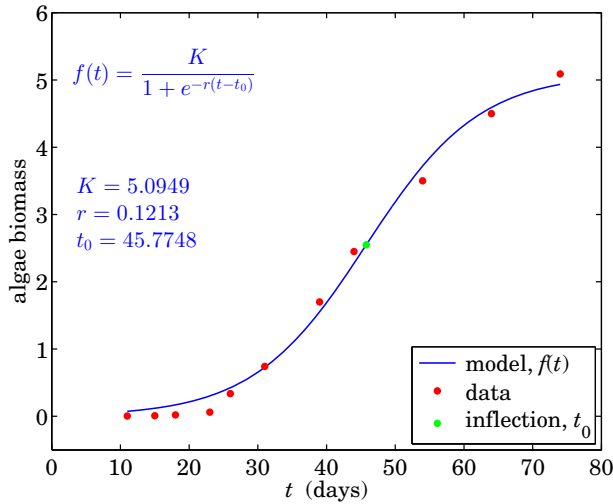
Growth Models



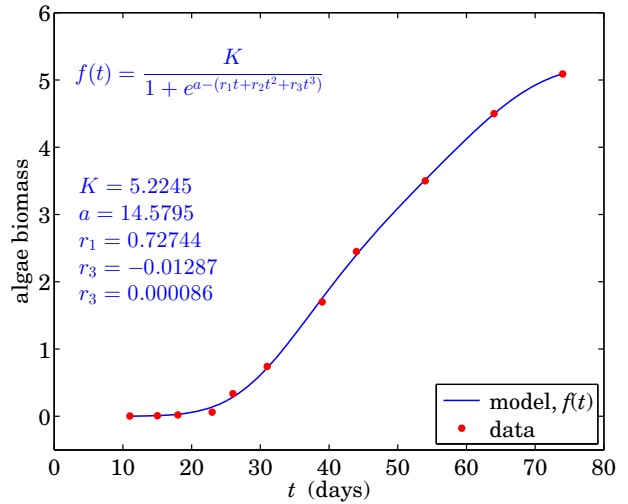
yeast growth – logistic model



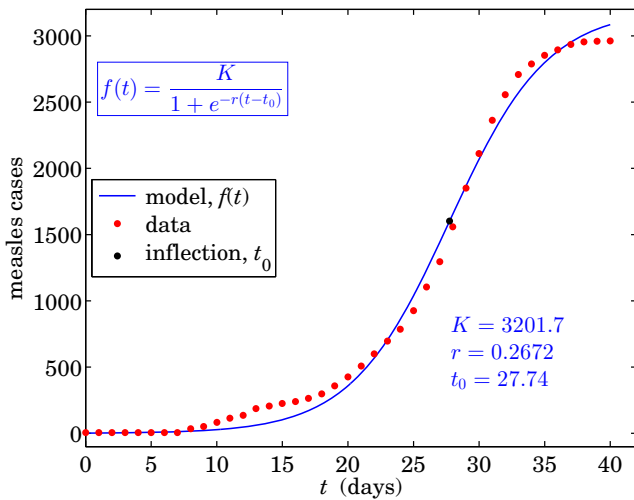
algae growth – logistic model



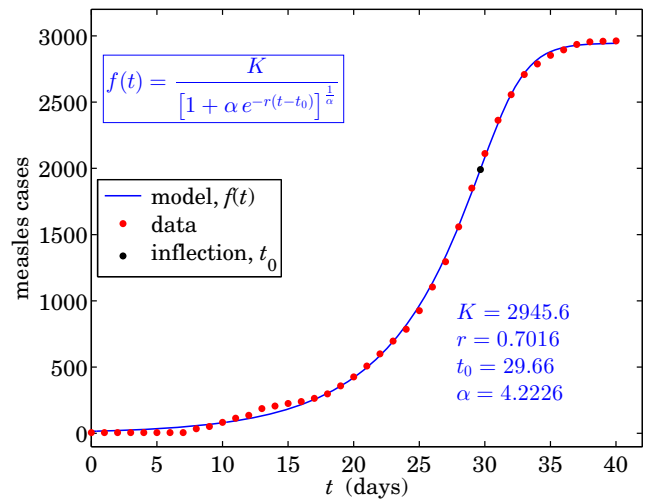
algae growth – Pearl–Reed model

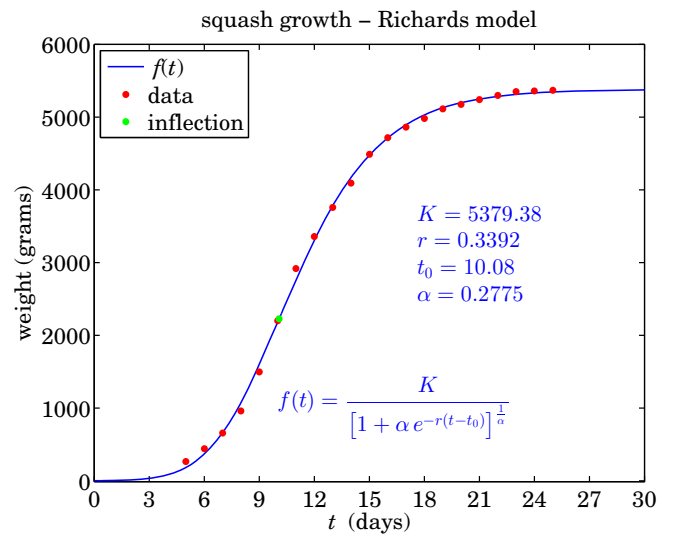
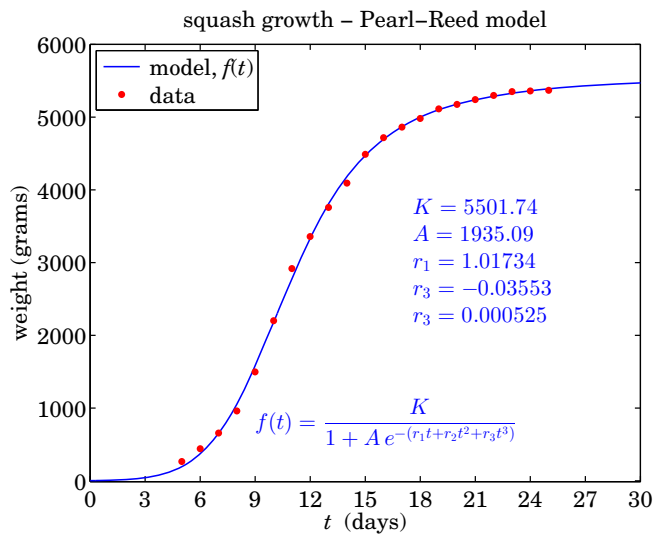
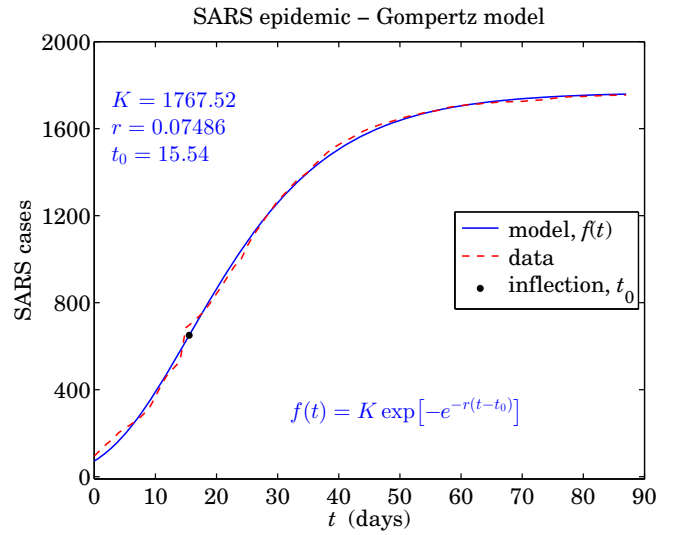
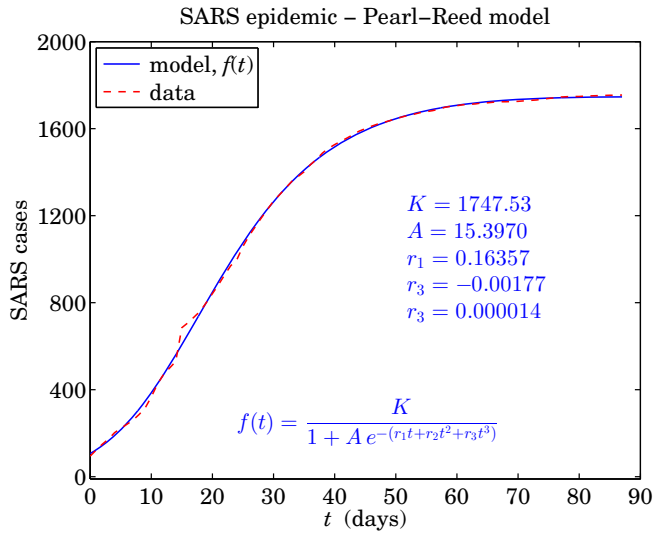
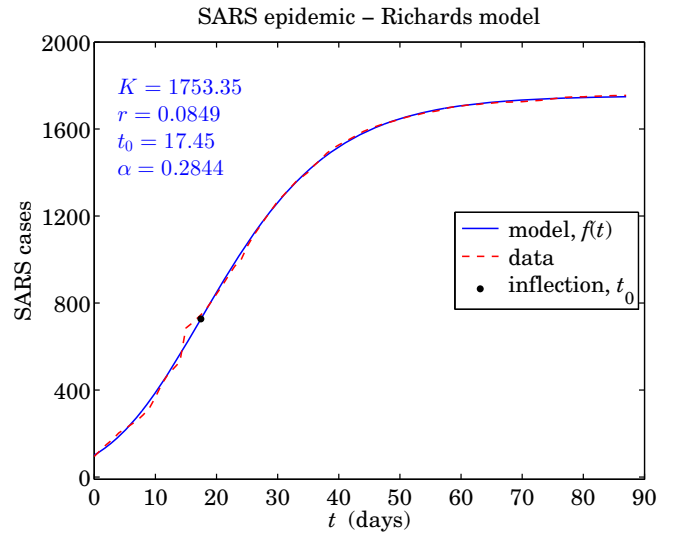
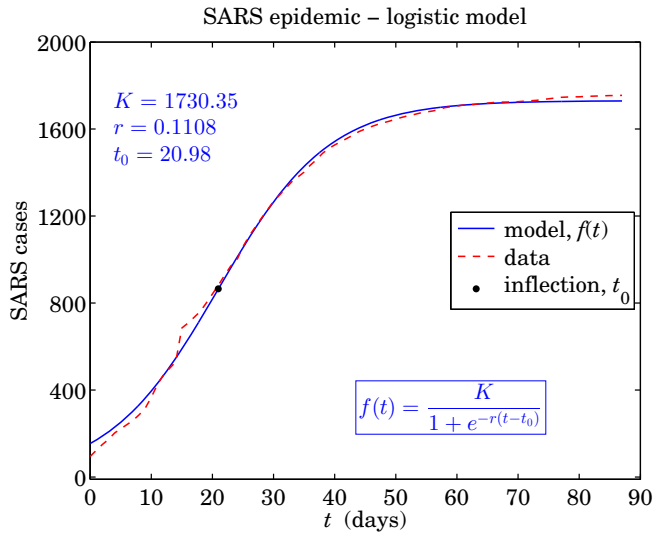


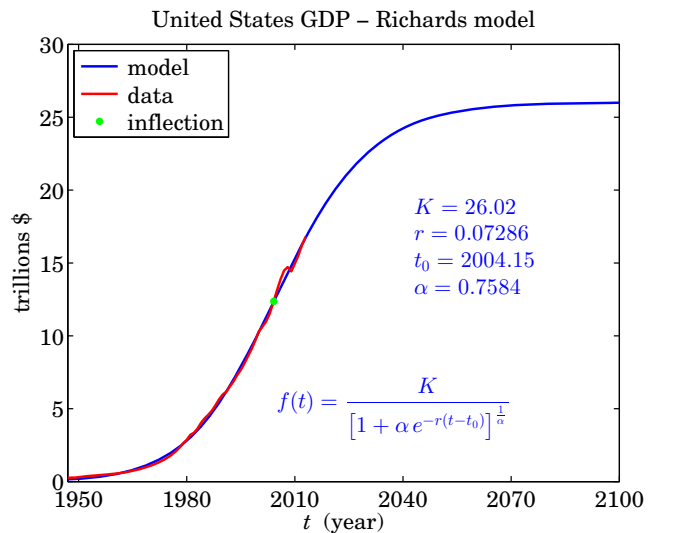
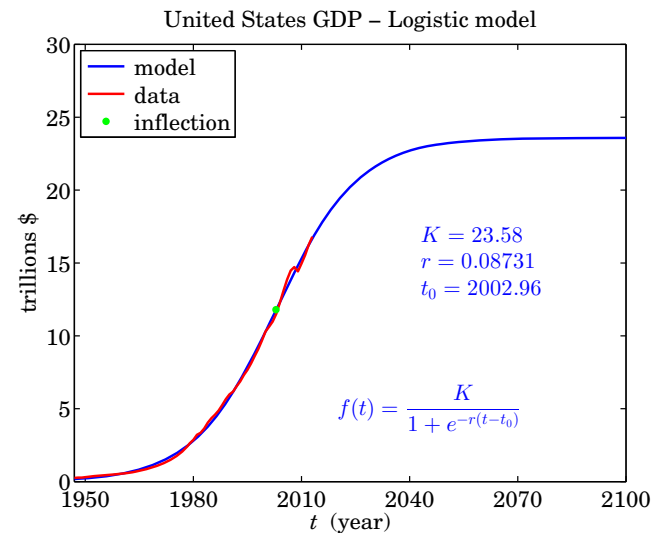
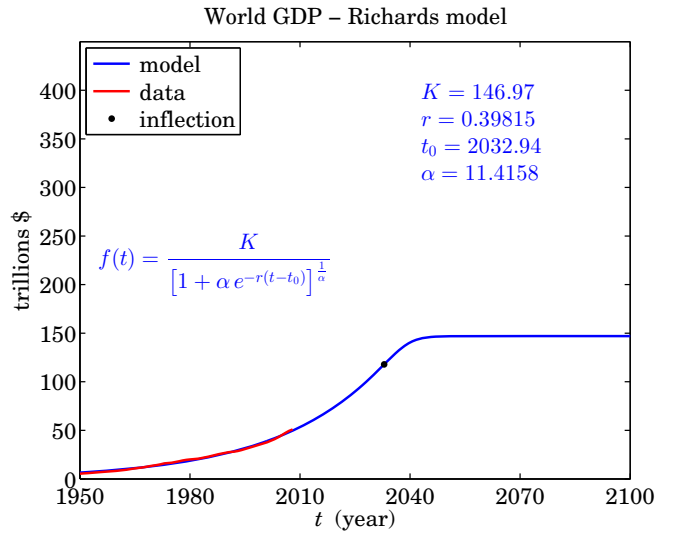
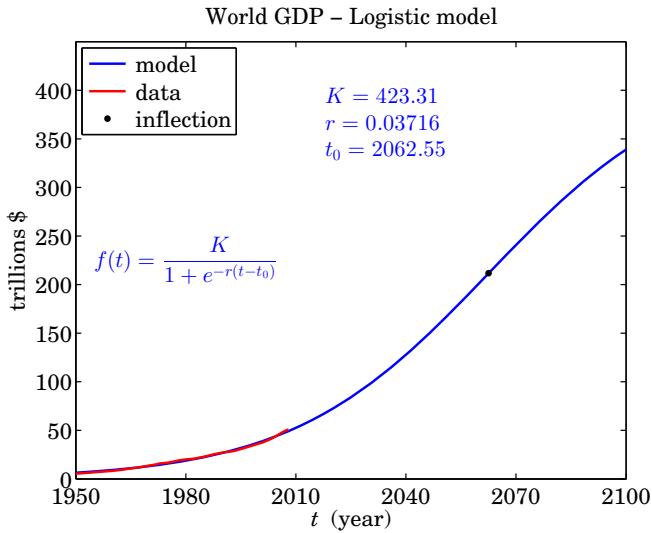
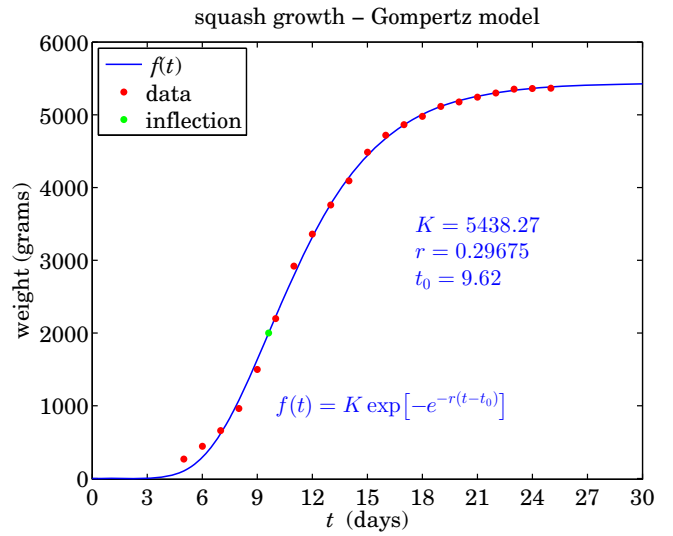
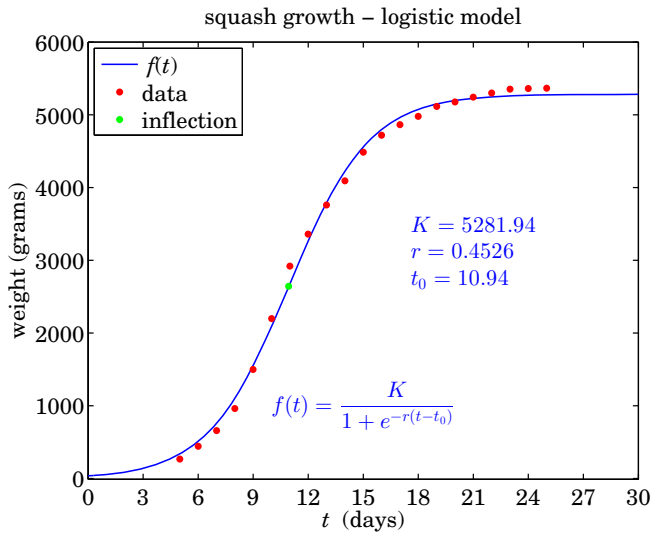
measles epidemic – logistic model



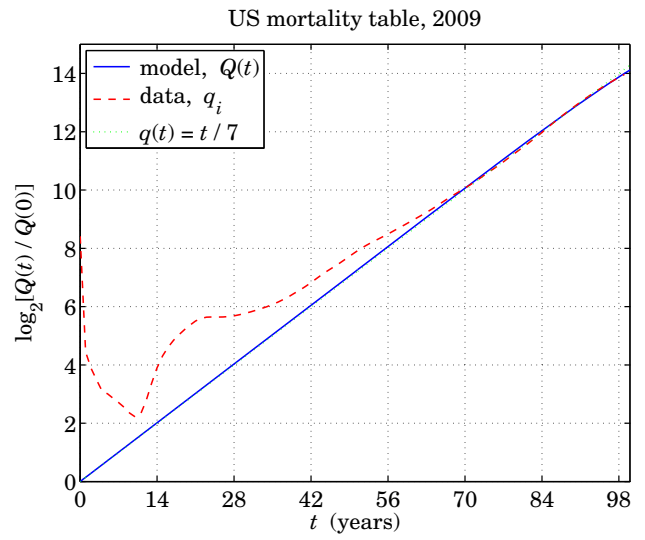
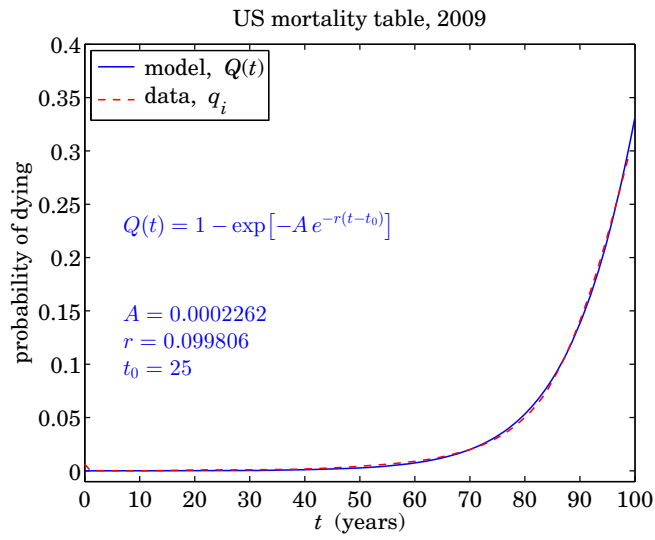
measles epidemic – Richards model







Life Tables



Homework Problems – Week 12
440:127 – Spring 2015 – S. J. Orfanidis

1. Consider the following incomplete set of temperature data for New York City (see the complete set in the file **NYCtemp.dat** on sakai),

	1971	1972	1973	1974	1975
jan	27.0		35.5		37.3
feb					
mar	40.1		46.4		40.2
apr					
may	61.4		59.5		65.8
jun					
jul	77.8		77.4		75.8
aug					
sep	71.6		69.5		64.2
oct					
nov					
dec	40.8		39.0		35.9

Construct the 6x3 matrix of given temperatures:

```
T = [27.0  35.5  37.3
      40.1  46.4  40.2
      61.4  59.5  65.8
      77.8  77.4  75.8
      71.6  69.5  64.2
      40.8  39.0  35.9];
```

- a. Using the function **interp2**, perform two-dimensional linear interpolation on the matrix *T* to fill in the values for the missing months and years. Use three **fprintf** commands to print the resulting interpolated table in the form:

	1971	1972	1973	1974	1975
jan	27.0	31.3	35.5	36.4	37.3
feb	33.5	37.3	41.0	39.9	38.8
mar	40.1	43.3	46.4	43.3	40.2
apr	50.8	51.9	53.0	53.0	53.0
may	61.4	60.5	59.5	62.6	65.8
jun	69.6	69.0	68.5	69.6	70.8
jul	77.8	77.6	77.4	76.6	75.8
aug	74.7	74.1	73.5	71.7	70.0
sep	71.6	70.5	69.5	66.8	64.2
oct	61.3	60.3	59.3	57.0	54.8
nov	51.1	50.1	49.2	47.3	45.3
dec	40.8	39.9	39.0	37.5	35.9

Plot the temperatures vs. months for one year, and also the temperatures vs. years for one month, and place also on the graphs the actual temperatures that you can obtain from the file **NYCtemp.dat** on sakai, see example graphs at the end.

- b. Perform the two-dimensional interpolation in two one-dimensional interpolation stages, first, applying **interp1** to fill the months of the three given years, and then, applying **interp1** again to that result, to fill the missing years. Print the result of the first stage using only three **fprintf** commands in the form:

	1971	1972	1973	1974	1975
jan	27.0		35.5		37.3
feb	33.5		41.0		38.8
mar	40.1		46.4		40.2
apr	50.8		53.0		53.0
may	61.4		59.5		65.8
jun	69.6		68.5		70.8
ju1	77.8		77.4		75.8
aug	74.7		73.5		70.0
sep	71.6		69.5		64.2
oct	61.3		59.3		54.8
nov	51.1		49.2		45.3
dec	40.8		39.0		35.9

c. Consider now a variation of this problem in which the actual missing temperatures are as shown below (the data are taken again from **NYCtemp.dat**), where the November data are given, but not the December ones,

	1971	1972	1973	1974	1975
jan	27.0		35.5		37.3
feb					
mar	40.1		46.4		40.2
apr					
may	61.4		59.5		65.8
jun					
ju1	77.8		77.4		75.8
aug					
sep	71.6		69.5		64.2
oct					
nov	45.1		48.3		52.3
dec					

Construct the corresponding matrix T of this data subset, and apply the **interp2** method to it and print the results using three **fprintf** commands. You will notice that the December data cannot be computed in the linear interpolation case because they lie outside the given range of months, and **interp2** produces NaN's for those values,

	1971	1972	1973	1974	1975
jan	27.0	31.3	35.5	36.4	37.3
feb	33.5	37.3	41.0	39.9	38.8
mar	40.1	43.3	46.4	43.3	40.2
apr	50.8	51.9	53.0	53.0	53.0
may	61.4	60.5	59.5	62.6	65.8
jun	69.6	69.0	68.5	69.6	70.8
ju1	77.8	77.6	77.4	76.6	75.8
aug	74.7	74.1	73.5	71.7	70.0
sep	71.6	70.5	69.5	66.8	64.2
oct	58.3	58.6	58.9	58.6	58.3
nov	45.1	46.7	48.3	50.3	52.3
dec	NaN	NaN	NaN	NaN	NaN

To handle this problem, we take advantage of the fact that there is annual periodicity in the data, and it does not really matter that we are numbering the months from 1-12 starting with January. Thus, we may append the January values as a 13th month below the missing December ones and perform linear interpolation on that subset,

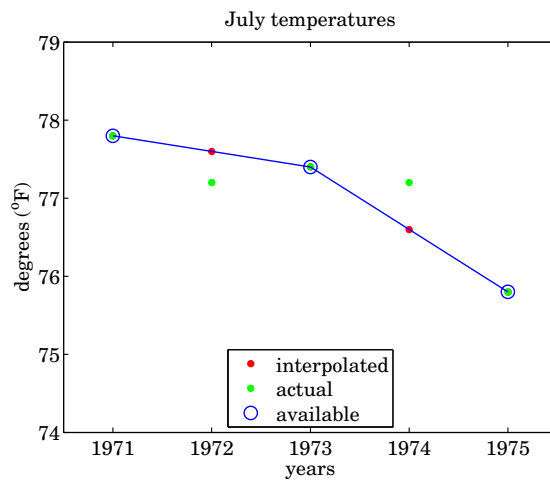
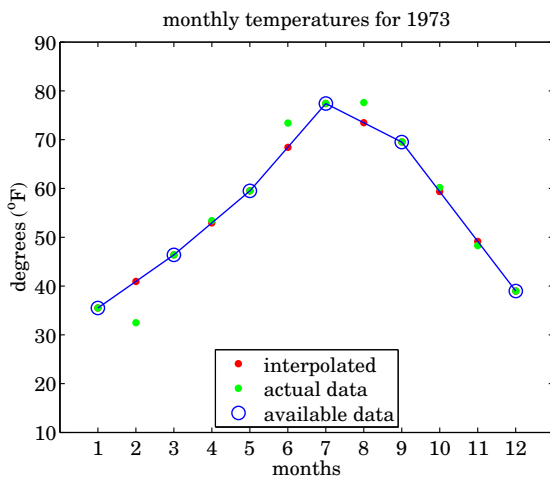
	1971	1972	1973	1974	1975
--	------	------	------	------	------

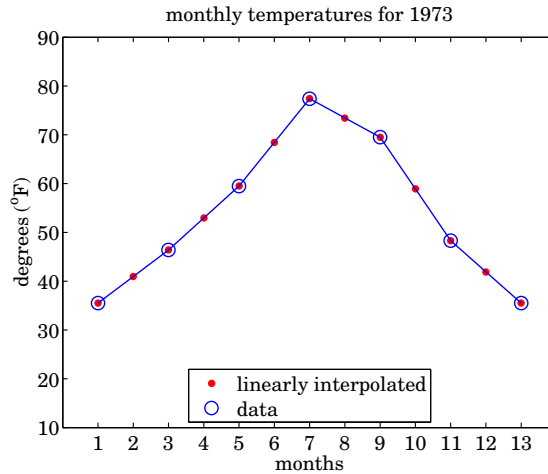
jan	27.0	35.5	37.3
feb			
mar	40.1	46.4	40.2
apr			
may	61.4	59.5	65.8
jun			
jul	77.8	77.4	75.8
aug			
sep	71.6	69.5	64.2
oct			
nov	45.1	48.3	52.3
dec			
jan	27.0	35.5	37.3

Determine the interpolated values using `interp2` and print the resulting table with three `fprintf` commands in the form:

	1971	1972	1973	1974	1975
jan	27.0	31.3	35.5	36.4	37.3
feb	33.5	37.3	41.0	39.9	38.8
mar	40.1	43.3	46.4	43.3	40.2
apr	50.8	51.9	53.0	53.0	53.0
may	61.4	60.5	59.5	62.6	65.8
jun	69.6	69.0	68.5	69.6	70.8
jul	77.8	77.6	77.4	76.6	75.8
aug	74.7	74.1	73.5	71.7	70.0
sep	71.6	70.5	69.5	66.8	64.2
oct	58.3	58.6	58.9	58.6	58.3
nov	45.1	46.7	48.3	50.3	52.3
dec	36.0	39.0	41.9	43.3	44.8
jan	27.0	31.3	35.5	36.4	37.3

Moreover, plot the interpolated monthly temperatures for 1973.





2. Consider the 5×5 matrix A where the $*$ entries are unknown:

$$A = \begin{bmatrix} 10 & * & 20 & * & 30 \\ * & * & * & * & * \\ 40 & * & 50 & * & 60 \\ * & * & * & * & * \\ 70 & * & 80 & * & 90 \end{bmatrix}$$

Determine estimates of the $*$ entries by linear interpolation using the following two methods.

- Using the function **interp2**.
- Using two calls to the function **interp1**.
- Repeat parts (a), (b) if the given numbers are viewed as part of a larger 7×7 matrix:

$$A = \begin{bmatrix} 10 & * & * & 20 & * & * & 30 \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ 40 & * & * & 50 & * & * & 60 \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ 70 & * & * & 80 & * & * & 90 \end{bmatrix}$$

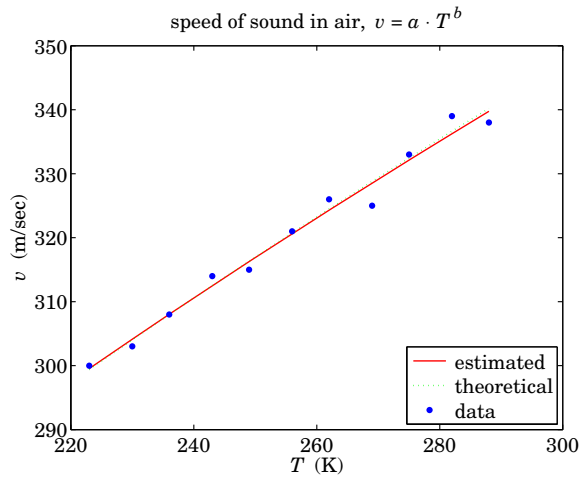
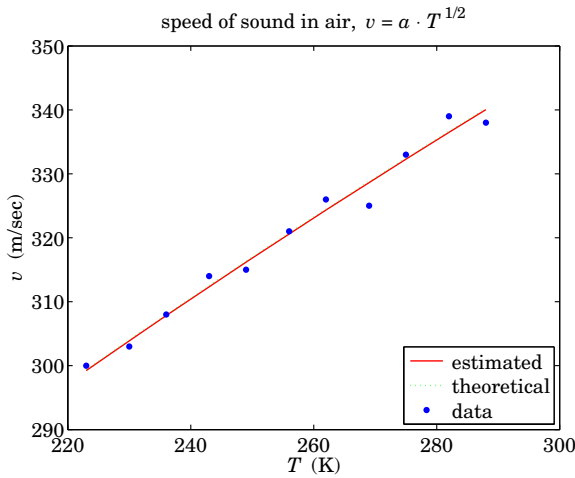
3. *Speed of Sound*. The speed of sound in air depends on the absolute temperature according the formula:

$$v_s = \sqrt{\gamma R_s T} = a \cdot \sqrt{T}, \quad a = \sqrt{\gamma R_s} = 20.0468 \quad (1)$$

where T is in Kelvins and v_s in m/sec, and $\gamma = 1.4$ and $R_s = 287.053$ is the specific gas constant of air in units of J/K/kg. The following measurements of v_s are given at 11 temperatures:

T_i	223	230	236	243	249	256	262	269	275	282	288	(K)
v_i	300	303	308	314	315	321	326	325	333	339	338	(m/sec)

- Using the model $v_s = a \cdot \sqrt{T}$, do a basis-function fit to this data to determine the parameter a and compare it with its theoretical value of $a = 20.0468$. Plot the estimated model over the range $220 \leq T \leq 290$, and add the theoretical model, and the data points to the graph.
- Repeat part (a) when the assumed model is $v_s = a \cdot T^b$, where both a, b are treated as unknown parameters to be estimated from the data.



4. *Air Density.* We saw in Week-9 Problem-6 that the following formula provided a good approximation to the density of air as a function of (geometrical) height up to about 40 km from sea level,

$$\rho(h) = 1.224 \cdot \exp \left[- \left(\frac{h}{11.66} \right) - \left(\frac{h}{18.19} \right)^2 + \left(\frac{h}{29.23} \right)^3 \right], \quad 0 \leq h \leq 40 \quad (2)$$

where ρ is in units of kg/m^3 and h in km. The following air density data, based on the US standard atmosphere model, can be loaded from the file `air.dat` on sakai,

h	rho	h	rho	h	rho
0	1.224999	14	0.227856	28	0.025076
2	1.006553	16	0.166471	30	0.018410
4	0.819346	18	0.121647	32	0.013555
6	0.660111	20	0.088910	34	0.009887
8	0.525786	22	0.064510	36	0.007258
10	0.413510	24	0.046938	38	0.005367
12	0.311938	26	0.034257	40	0.003996

Consider the following three possible models:

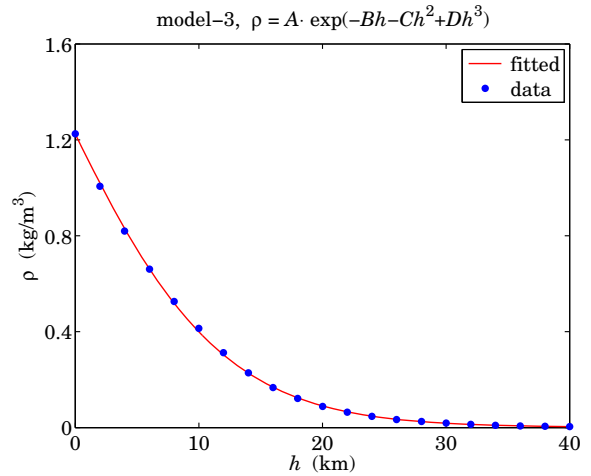
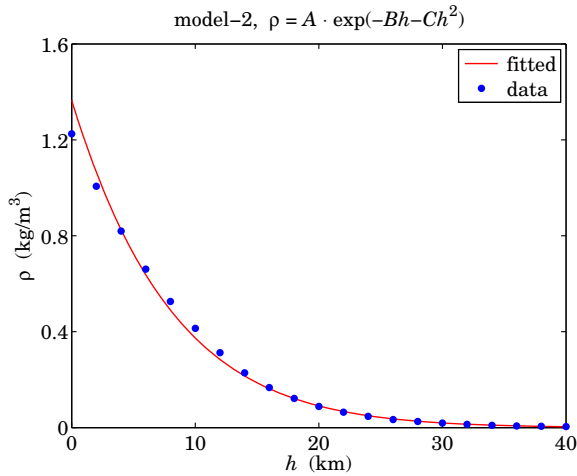
model-1: $\rho = A \cdot e^{-Bh}$

model-2: $\rho = A \cdot e^{-Bh - Ch^2}$ (3)

model-3: $\rho = A \cdot e^{-Bh - Ch^2 + Dh^3}$

Using the basis-functions method (do not use polyfit), fit the given data to each model and in each case determine the model parameters A, B, C, D and make a plot of the model versus height in the range $0 \leq h \leq 40$ km, and add the data points to the graphs. Put model-3 in the following form, determine the parameters ρ_0, h_1, h_2, h_3 , and compare them with those of Eq. (2),

$$\rho(h) = \rho_0 \cdot \exp \left[- \left(\frac{h}{h_1} \right) - \left(\frac{h}{h_2} \right)^2 + \left(\frac{h}{h_3} \right)^3 \right], \quad 0 \leq h \leq 40$$



5. *CO₂ concentration.* At the end of the Week-12 lecture notes, the following model is used to fit CO₂ concentrations for the 33-year period from Jan 1980 to Dec 2012, where t is in units of months,

$$y = \underbrace{c_0 + c_1 t + c_2 t^2 + c_3 t^3}_{\text{trend component}} + \underbrace{c_4 \cos\left(\frac{2\pi t}{12}\right) + c_5 \cos\left(\frac{2\pi t}{12}\right)}_{\text{cyclical component}} \quad (4)$$

The data can be loaded from the file **co2.dat** on sakai and downloaded from:

ftp://ftp.cmdl.noaa.gov/ccg/co2/trends/co2_mm_g1.txt
<http://www.esrl.noaa.gov/gmd/ccgg/trends/>

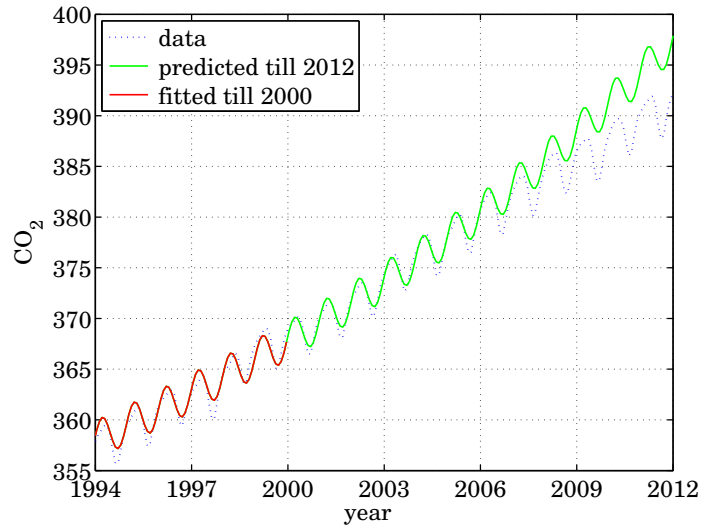
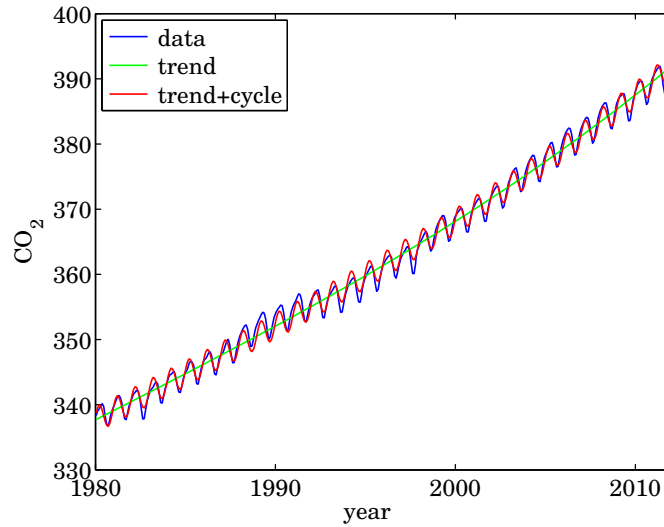
- a. Using the basis-function method, determine the coefficients c_i using the entire 33-year data set. Plot the fitted curve, the trend, and the actual data versus year. Note that the above data set defines the time samples to lie in the middle of each month, so that for the purpose of plotting, the time in units of a fraction of a year is given by

$$t_y = \frac{1}{12} \left(t + \frac{1}{2} \right) + 1980, \quad t = 0, 1, 2, \dots, 395$$

where we note that there are 396 months in 33 years.

- b. Next, we look at the predictive ability of this model. Repeat the data fitting of part (a), but now use only the 20-year period from Jan 1980 to Dec 1999. Then, use the estimated coefficients c_i to predict the subsequent years from Jan 2000 to Dec 2012.

On the same graph, plot the actual data for the years 1997-2012, add to the graph the fitted curve until 2000, and add the predicted curve for the remaining years of 2000-2012. See example graph at the end.



6. *Cosmic Microwave Background*. This is a continuation of the CMB spectrum example from the week-11 lecture notes. Here, we explore different optimization criteria for estimating the CMB temperature. We recall that the Planck spectrum is given as follows, where f is in GHz and I in units of mega-Jansky per steradian, MJy/sr:[†]

$$I(T, f) = \frac{Af^3}{\exp\left(\frac{Bf}{T}\right) - 1} \quad (5)$$

with the following values of the parameters:

$$\begin{aligned} h &= 6.62606957 \cdot 10^{-34}, & \text{J/Hz, Planck's constant} \\ c &= 2.99792458 \cdot 10^8, & \text{m/s, speed of light, } 29.9792458 \text{ cm} \cdot \text{GHz} \\ k &= 1.3806488 \cdot 10^{-23}, & \text{J/K, Boltzmann constant} \\ A &= 10^{47} \frac{2h}{c^2}, & B = 10^9 \frac{h}{k} \end{aligned}$$

[†] 1 GHz = 10^9 Hz, 1 Jy = $10^{-26} \frac{\text{W}}{\text{m}^2 \cdot \text{Hz}}$.

- a. The data file on sakai, **firas.dat**, contains $N = 43$ experimental values of frequencies f_i and radiances I_i , $i = 1, 2, \dots, N$. Load this data file into Matlab and convert the frequencies f_i in units of GHz as discussed in the week-11 powerpoints.[‡]
- b. Define an anonymous function $I(f, T)$ of two independent variables T, f implementing Eq. (5). The function must be vectorized in the variable f , but not in T .
- c. Given a value of T , let $e_i(T) = I_i - I(T, f_i)$, $i = 1, 2, \dots, N$ be the observation error residuals. The basic least-squares error criterion attempts to minimize the sum of the squares of $e_i(T)$ with respect to T , that is, T is chosen so as to minimize the function:

$$E(T) = \sum_{i=1}^N |e_i(T)|^2 = \sum_{i=1}^N |I_i - I(T, f_i)|^2 = \min, \quad (\text{least-squares criterion}) \quad (6)$$

Define the function $E(T)$ in MATLAB and minimize it using, **fminbnd**, by searching in the temperature interval, $1 \leq T \leq 3$ K. Recall from the lecture notes that it can be implemented very easily in MATLAB with the help of the anonymous function of part (b):

$$E = @(T) \text{sum}((Ii-I(T, fi)).^2);$$

The built-in function, **nlinfit**, also uses the same criterion Eq. (6), but it employs a different minimization method (the Levenberg-Marquardt algorithm) than **fminbnd**. Compare the results from these two methods.

- d. Consider the following alternative minimization criteria that are often used in practice:

$$E(T) = \sqrt{\sum_{i=1}^N |e_i(T)|^2} = \min, \quad (L_2 \text{ norm})$$

$$E(T) = \sum_{i=1}^N |e_i(T)| = \min, \quad (L_1 \text{ norm}) \quad (7)$$

$$E(T) = \max\{|e_1(T)|, |e_2(T)|, \dots, |e_N(T)|\} = \min, \quad (L_\infty \text{ norm})$$

Implement these criteria in a vectorized form in MATLAB and determine the estimated T based on each one of them.

- e. If the i th measurement were exact, one would have the following condition, which can be solved for the temperature T :

$$I_i = I(T, f_i) = \frac{Af_i^3}{\exp\left(\frac{Bf_i}{T}\right) - 1} \Rightarrow T = \frac{f_i}{g_i}, \quad g_i = \frac{1}{B} \log\left(1 + \frac{Af_i^3}{I_i}\right) \quad (8)$$

This leads to the following two possible least-squares criteria with the explicit solutions:

$$E(T) = \sum_{i=1}^N \left(T - \frac{f_i}{g_i}\right)^2 = \min \Rightarrow T = \frac{1}{N} \sum_{i=1}^N \frac{f_i}{g_i} \quad (9)$$

$$E(T) = \sum_{i=1}^N (g_i T - f_i)^2 = \min \Rightarrow T = \frac{\sum_{i=1}^N f_i g_i}{\sum_{i=1}^N g_i^2} \quad (10)$$

Define g_i as a column vector in terms of the column vectors f_i, I_i . Then, using single vectorized commands, calculate T from Eqs. (9) and (10).

[‡]the first column contains the wavenumbers, or inverse wavelengths λ^{-1} in units of cm^{-1} , which can be converted to GHz using the formula $f = c\lambda^{-1}$, with $c = 29.9792458 \text{ cm} \cdot \text{GHz}$.

- f. The result of Eq. (8) may be thought of as defining the following two over-determined linear systems of equations in the single unknown T :

$$\begin{aligned} \text{system 1: } \quad T &= \frac{f_i}{g_i}, \quad i = 1, 2, \dots, N \\ \text{system 2: } \quad g_i T &= f_i, \quad i = 1, 2, \dots, N \end{aligned} \tag{11}$$

or, written vectorially,

$$\begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} T = \begin{bmatrix} f_1/g_1 \\ \vdots \\ f_N/g_N \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} g_1 \\ \vdots \\ g_N \end{bmatrix} T = \begin{bmatrix} f_1 \\ \vdots \\ f_N \end{bmatrix}$$

We may think of these as special cases of the basis-functions method, with basis T . Show how to obtain the MATLAB solution of these systems using the backslash operator. Note, that the results are equivalent to Eqs. (9) and (10).

- g. Collect together the seven different estimates of T obtained in parts (c-e), and using at most *three* **fprintf** commands print the results exactly as in the following table (you will need to convert all items to be printed into cell arrays):

method	T
least-squares	2.725015
nlinfit	2.725013
L2 norm	2.725003
L1 norm	2.725027
L_inf norm	2.724235
system 1	2.724458
system 2	2.723501

- h. The above methods regarded $I(T, f)$ as a function of a single parameter T to be fitted. However, it is possible to regard it as a function of both the parameters A, T and estimate both from the measured data. Consider the two-parameter vector b and function $F(b, f)$:

$$F(b, f) = \frac{b(1)f^3}{\exp\left(\frac{Bf}{b(2)}\right) - 1}, \quad b = \begin{bmatrix} A \\ T \end{bmatrix}$$

Pass this function into **nlinfit** with the initial choice $b_0 = [0.001; 3]$ and estimate both parameters A, T . Using only three **fprintf** commands, make a table like the one below, comparing the values of A, T from the original 1-parameter fit and the present 2-parameter fit:

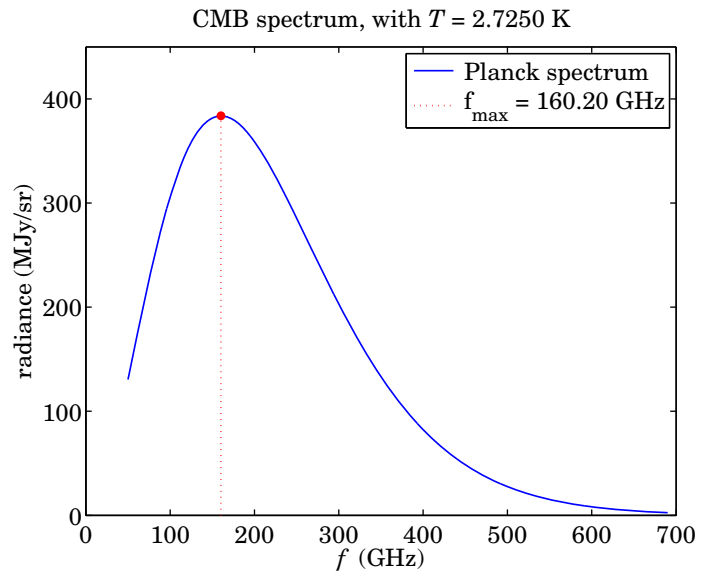
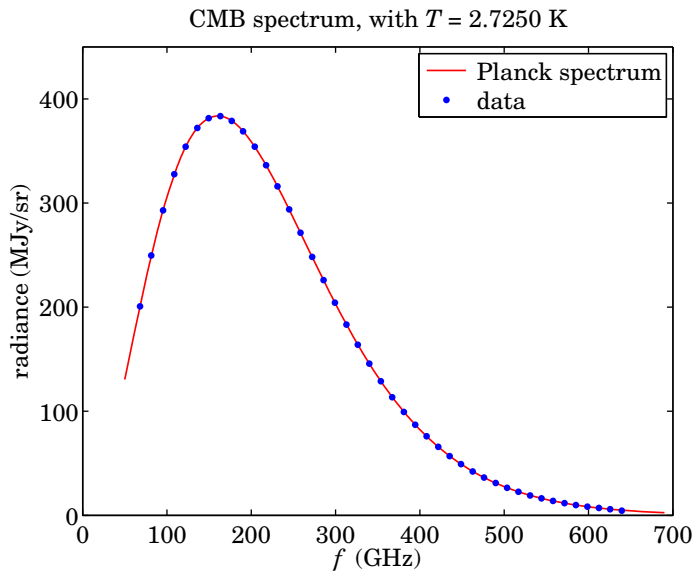
method	A	T
1-parameter fit	0.00147450	2.725013
2-parameter fit	0.00147454	2.724992

- i. Using the value of T from the least-squares method of part (c), calculate and plot the spectral radiance of Eq. (5) over the frequency interval, $50 \leq f \leq 690$ GHz, and add the measured data points f_i, I_i to the graph.

Using the function **fminbnd**, calculate the maximum of the function $I(T, f)$ and the frequency at which it occurs, and on a separate graph plot again $I(T, f)$ and add the maximum point to the graph. See example graphs at end.

CMB References – PDFs on Sakai

1. T. J. Pfaff, et al., “The Use of Statistics in Experimental Physics,” *Math. Mag.*, **86**, 120 (2013).
2. D. J. Fixsen, et al., “The Cosmic Microwave Background Spectrum from the Full COBE FIRAS Data Set,” *Astrophys. J.*, **473**, 576 (1996).
3. 2006 Nobel Prize in Physics to J. C. Mather and G. F. Smoot, <http://www.kva.se/en/pressroom/press-releases-2008-2001/The-Nobel-Prize-in-Physics-2006/>, and PDF file, http://www.kva.se/Documents/Priser/Nobel/2006/pop_fy_en_06.pdf.
4. J. C. Mather, “Nobel Lecture: From the Big Bang to the Nobel Prize and beyond,” *Rev. Mod. Phys.*, **79**, 1331 (2007).
5. G. F. Smoot, “Nobel Lecture: Cosmic microwave background radiation anisotropies: Their discovery and utilization,” *Rev. Mod. Phys.*, **79**, 1349 (2007).
6. A. A. Penzias, “The origin of the elements,” Nobel lecture, *Rev. Mod. Phys.*, **51**, 425 (1979).
7. R. W. Wilson, “The cosmic microwave background radiation,” Nobel lecture, *Rev. Mod. Phys.*, **51**, 433 (1979).
8. V. S. Alpher, “Ralph A. Alpher, Robert C. Herman, and the Cosmic Microwave Background Radiation,” *Phys. Perspect.*, **14**, 300 (2012).
9. P. J. E. Peebles, “Discovery of the Hot Big Bang: What happened in 1948,” Oct. 2013, <http://arxiv.org/abs/1310.2146>.



7. *Bass Diffusion Model.* The Bass model is the most influential model in marketing engineering for describing the *diffusion of innovations*, that is, the introduction, acceptance, adoption, growth, and eventual decline of new products, ideas, innovations, techniques, services, and procedures.

It is characterized by three parameters, m, p, q , where m is the maximum number of potential adopters or buyers of a product, and q, p represent growth rates of the so-called “imitators” and “innovators”, where the former are influenced by the number of existing adaptors, while the latter are not and decide independently of others to adopt a product. The mathematical form of the model and the differential equation from which it arises are as follows:

$$\boxed{N(t) = m \frac{1 - e^{-(p+q)t}}{1 + \frac{q}{p} e^{-(p+q)t}}, \quad \frac{dN(t)}{dt} = \underbrace{p[m - N(t)]}_{\text{innovators}} + \underbrace{qN(t) \left[1 - \frac{N(t)}{m}\right]}_{\text{imitators}}} \quad (12)$$

where $t \geq 0$ measures the years since the introduction of the product, and $N(t)$ is the cumulative number of adaptors up to time t , and dN/dt represents the rate of adoption. Initially, we have $N(0) = 0$ at $t = 0$. The function $N(t)$ is an S-shaped curve resembling the logistic curve (see example graphs at the end and Problem 12 on the week-11 homework set). In fact, if Eq. (12) is modified to allow a nonzero initial value $N(0)$, then Eq. (12) becomes the logistic model in the limit $p = 0$. In practice, we typically have $q \gg p$. More generally, the solution relating $N(t)$ and $N(t_1)$ at an earlier time instant is:

$$N(t) = m \frac{[q + p e^{-(p+q)(t-t_1)}]N(t_1) + mp[1 - e^{-(p+q)(t-t_1)}]}{q[1 - e^{-(p+q)(t-t_1)}]N(t_1) + m[p + q e^{-(p+q)(t-t_1)}} \quad (13)$$

The derivative dN/dt is a bell-shaped curve and has a maximum corresponding to the inflection time t_0 of $N(t)$, as follows:

$$t_0 = \frac{1}{q+p} \ln\left(\frac{q}{p}\right), \quad N(t_0) = m \frac{q-p}{2q}, \quad \frac{dN(t_0)}{dt} = m \frac{(q+p)^2}{4q} \quad (14)$$

Eqs. (14) are very important to manufacturers because they determine the time $t = t_0$ of maximum rate of adoption of a product, to be followed by its subsequent decline at $t > t_0$.

The model parameters m, p, q are determined by fitting the model to the observed sales data. There exist several fitting methods, but here, we will use the least-squares method implemented by **nlinfit**. This method requires that we provide initial estimates of the parameters, say, m_0, p_0, q_0 . For m_0 , we may choose the maximum observed value of $N(t)$. And if we have estimates, say N_0, \dot{N}_0 of $N(t_0)$ and $dN(t_0)/dt$, then we may solve Eqs. (14) to obtain initial estimates of p, q ,

$$N_0 = m_0 \frac{q_0 - p_0}{2q_0}, \quad \dot{N}_0 = m_0 \frac{(q_0 + p_0)^2}{4q_0} \Rightarrow q_0 = \frac{m_0 \dot{N}_0}{(m_0 - N_0)^2}, \quad p_0 = \frac{\dot{N}_0 (m_0 - 2N_0)}{(m_0 - N_0)^2} \quad (15)$$

Given column vectors of, say, K observed data pairs $\{t_i, N_i, i = 1, 2, \dots, K\}$, we can define the model function of Eq. (12) as an anonymous MATLAB function of the parameters m, p, q and t , then define initial estimates of the parameters, and obtain the solution as follows, where the three parameters m, p, q are represented by the three components $c(1), c(2), c(3)$ of the parameter vector c ,

```
f = @(c,t) c(1) *(1 - exp(-(c(2)+c(3))*t))./(1 + c(3)/c(2)*exp(-(c(2)+c(3))*t));

[Nd0,i0] = max(diff(Ni)./diff(ti)); % estimate the maximum of the derivative
N0 = Ni(i0); % estimate the value at inflection
m0 = max(Ni); % estimate the maximum level
q0 = m0*Nd0/(m0-N0)^2; % initial q and p
p0 = Nd0*abs(m0-2*N0)/(m0-N0)^2; % note the absolute value

c0 = [m0, p0, q0]'; % initial parameter vector

c = nlinfit(ti,Ni,f,c0); % fitted parameters
```

The data file, **bass.dat**, contains measured sales data for seven products: air conditioners, color televisions, clothes dryers, ultrasound equipment, mammography facilities in hospitals, foreign language schools, and accelerated learning programs. The data are from Ref. [2] below and have served as benchmarks for evaluating different estimation methods of various diffusion models, including the Bass model. Logistic models and their relatives, such as Richards and Gompertz models, can also be used as diffusion of innovations models. See Ref. [5] below for a comprehensive review.

- a. Load the data file into MATLAB and, for each of the above seven cases, extract the corresponding time and data vectors t_i, N_i . Because some cases have different time lengths, the data file has been padded with NaNs to fill the blanks, so that it can be loaded as a whole into MATLAB. Before proceeding with each case, you must remove any potential NaNs using, for example, the code,

```
Ni = Ni(~isnan(Ni));           % remove possible NaNs in some cases
ti = ti(~isnan(ti));
ti = ti-ti(1)+1;              % define relative time, ti = 1,2,3,...
```

Determine the model parameters m, p, q for each case by following the above estimation procedure based on **nlinfit**. Make a plot of the estimated model function $f(c, t)$ vs. t in the interval $0 \leq t \leq 15$, and add the data points on the graph, and also indicate the inflection point. The results of this estimation method are virtually identical to those of Ref. [6] below. Print your results as follows:

product	m	p	q
air conditioners	17.17	0.007439	0.426984
color televisions	38.31	0.016416	0.655472
clothes dryers	15.42	0.012171	0.360685
ultrasound devices	204.99	0.005830	0.423133
mammography hospitals	125.37	0.002290	0.651793
language schools	42.65	0.004871	0.549531
learning programs	65.94	0.000982	0.879754

- b. Instead using the built-in function **nlinfit**, try a do-it-yourself approach that uses the built-in function **fminsearch** to directly minimize the sum of squared errors. A similar approach was used in the previous problem on the CMB. Given the K data pairs t_i, N_i and the model function $f(c, t)$ defined above, the least-squares method minimizes the following function of the parameter vector c :

$$J(c) = \sum_{i=1}^K [N_i - f(c, t_i)]^2 = \min \quad (16)$$

This function and its minimization using **fminsearch** can be easily implemented in MATLAB, assuming that $f(c, t)$ has already been defined and an initial search vector c_0 has been chosen:

```
J = @(c) sum((Ni - f(c,ti)).^2);   % define least-squares function
c = fminsearch(J,c0);             % find c that minimizes J(c)
```

Carry out this minimization procedure for each of the seven cases and compare your results to those obtained in part (a).

Diffusion Models – PDFs on Sakai

1. F. M. Bass, “A New Product Growth Model for Consumer Durables,” *Management Sci.*, 15(5), 215 (1969). <http://www.jstor.org/stable/2628128>
2. V. Mahajan, C.H. Mason, and V. Srinivasan, “An Evaluation of Estimation Procedures for New Product Diffusion Models,” in V. Mahajan, Y. Wind (eds.), *Innovation Diffusion Models of New Product Acceptance*, Ballinger Publishing Company, Cambridge, 1986, pp. 203. <https://gsbapps.stanford.edu/researchpapers/library/RP851.pdf>
3. F. M. Bass, Comments on “A New Product Growth for Model Consumer Durables”: The Bass Model, *Management Sci.*, 50(12), 1833 (2004). <http://www.jstor.org/stable/30046154>

4. R. Peres, E. Muller, and V. Mahajan, "Innovation Diffusion and New Product Growth Models: A Critical Review and Research Directions," *Intern. J. of Research in Marketing*, 27 91 (2010).
5. N. Meade and T. Islam, "Modelling and Forecasting the Diffusion of Innovation—A 25-year Review," *Intern. J. Forecasting*, 22, 519 (2006).
6. D. Jukić, "On Nonlinear Weighted Least Squares Estimation of Bass Diffusion Model," *Appl. Math. Comput.*, 219, 7891 (2013).

