

# Math 541 - Numerical Analysis

## Lecture Notes – MatLab Programming

Joseph M. Mahaffy,  
(`jmahaffy@mail.sdsu.edu`)

Department of Mathematics and Statistics  
Dynamical Systems Group  
Computational Sciences Research Center  
San Diego State University  
San Diego, CA 92182-7720

<http://jmahaffy.sdsu.edu>

Spring 2018

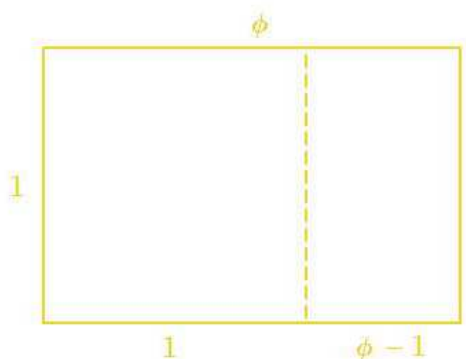
# Outline

- 1 Golden Ratio and MatLab
  - MatLab Basics
  - MatLab Script
  - MatLab Function
- 2 Fibonacci Numbers
  - MatLab Fibonacci
  - Fibonacci and Golden Ratio
- 3 Collatz Problem
- 4 MatLab Programming and Series
  - Exponential Function,  $e^x$
  - Graph of  $e^x$
  - Bessel Functions

# Golden Rectangle

Since ancient times, the **Golden Rectangle** has been believed to be aesthetically pleasing, where the **large rectangle** ( $\phi \times 1$ ) composed of a **unit square** and **smaller rectangle** is *similar* to the adjacent **smaller rectangle** ( $1 \times (\phi - 1)$ )

The **Golden Rectangle** is shown below



## Golden Ratio

The **Golden Ratio** satisfies

$$\frac{1}{\phi} = \frac{\phi - 1}{1}$$

It follows that  $\phi$  satisfies

$$\phi^2 - \phi - 1 = 0,$$

which by the quadratic equation gives

$$\phi = \frac{1 \pm \sqrt{5}}{2}$$

Since only the positive root makes sense

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

# Golden Ratio and MatLab

**MatLab:** We explore MatLab commands

- Numbers in MatLab:

```
phi = (1 + sqrt(5)) / 2
```

- Default is 4 places after decimal
- Obtain 16 digits (**MatLab** standard) with  
`format long` or `double(phi)`
- Obtain 50 digits with  
`vpa(phi, 50)`
- The polynomial is  $x^2 - x - 1$ , which is solved by  

```
p = [1 -1 -1]  
r = roots(p)
```
- Symbolic package to solve quadratic:  

```
syms x  
r = solve(1/x == x-1)
```

## Basic MatLab – Function and Graph

**MatLab:** Define a basic function and study

- Create an *anonymous function*

```
f = @(x) 1./x-(x-1)
```

- Create a simple plot of the function: `ezplot(f, 0, 4)`  
Note the plot selects a default reasonable range

- Find the zero of a function

```
phi = fzero(f, 1)
```

- Add this point to the graph

```
hold on
```

```
plot(phi, 0, 'o')
```

- Use the `fsolve` routine from the Optimization package

```
fsolve(@(x) f(x), 1)
```

# MatLab Script for Graph

Create **MatLab Scripts** to make graphs (modify old ones)

```
1   % GOLDRECT Plot the golden rectangle
2
3   phi = (1 + sqrt(5))/2;
4   x = [0 phi phi 0 0];
5   y = [0 0 1 1 0];
6   u = [1 1];
7   v = [0 1];
8   plot(x,y,'b',u,v,'b--')
9   text(phi/2, 1.05, '$\phi$', 'interpreter', 'latex')
10  text((1+phi)/2, -0.05, '$\phi - ...
    1$', 'interpreter', 'latex')
11  text(-0.05,0.5, '1')
12  text(0.5,-0.05, '1')
13  axis equal
14  axis off
15  set(gcf, 'color', 'white')
```

## MatLab Function

The **Golden fraction** is a continued fraction of the form:

$$\phi = 1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \dots}}}}$$

- Create a **MatLab Function** that produce this continued fraction to  $n$  terms
- Display the continued fraction to  $n$  terms
- Compute the fractional value, then show the decimal value of this
- Find the error of truncating after  $n$  terms



# MatLab Function

```
1 function goldfrac(n)
2 %GOLDFRACT Golden ratio continued fraction
3 % GOLDFRAC(n) displays n terms
4 p = '1';
5 for k = 1:n
6     p = ['1+1/(' p ')'];
7 end
8 p
9 p = 1; q = 1;
10 for k = 1:n
11     s = p;
12     p = p + q;
13     q = s;
14 end
15 p = sprintf('%d/%d',p,q)
16 format long; p = eval(p)
17 format short; err = (1 + sqrt(5))/2 - p
18 end
```

# Fibonacci Numbers

**Fibonacci Numbers** appear often in **Nature** ([hyperlink](#)), petals of flowers, organization of pine cones and sunflowers, branching trees and bones, shell spirals, reproduction, etc.

Leonardo Fibonacci originally posed the following:

*A man put a pair of rabbits in a place surrounded on all sides by a wall. How many pairs of rabbits can be produced from that pair in a year if it is supposed that every month each pair begets a new pair which from the second month on becomes productive?*

Malthusian growth, where immediate fertility is assumed, gives the **difference equation**

$$P_{n+1} = (1 + r)P_n,$$

which is easily solved, given an initial population,  $P_0$

## Fibonacci Numbers

Fibonacci's problem introduces a delay for maturation, complicating the problem

If  $f_n$  denotes the number of pairs of rabbits after  $n$  months, then the number of pairs is the number at the beginning plus the number of births or the **difference equation** becomes:

$$f_n = f_{n-1} + f_{n-2}$$

with initial conditions,  $f_0 = 1$  and  $f_1 = 1$ , so

$$f_2 = f_1 + f_0 = 1 + 1 = 2$$

and

$$f_3 = f_2 + f_1 = 2 + 1 = 3, \quad f_4 = f_3 + f_2 = 3 + 2 = 5$$

## Fibonacci Program

We assume that we start with **one pair** of rabbits ( $f_0 = 1$ )  
They don't reproduce the first month, so there is still one pair  
( $f_1 = 1$ )

We write a MatLab program for the **Fibonacci sequence**

```
1 function f = fibonacci(n)
2 %FIBONACCI: Fibonacci sequence
3 %   f = FIBONACCI(n) generates the first n ...
   Fibonacci numbers
4 f = zeros(n,1); % (n x 1) matrix of zeros
5 f(1) = 1;
6 f(2) = 2;
7 for k = 3:n
8     f(k) = f(k-1) + f(k-2);
9 end
10 end
```

## Fibonacci vs Malthusian Growth

**Fibonacci** asked how many pairs of rabbits after 12 months, so invoking the program:

```
fibonacci(12)
```

shows the sequence

```
1 2 3 5 8 13 21 34 55 89 144 233
```

**Malthusian growth** doubles every month, so MatLab computes with

```
2.^(1:12)
```

resulting in

```
2 4 8 16 32 64 128 256 512 1024 2048 4096
```

It follows that the delay in maturation results in 233 pairs of rabbits at the end of the year, compared to 4096 without the maturation

## Fibonacci Program

The program below uses a `while` loop to determine how long it takes to pass a given number in a Fibonacci sequence

```
1 function f = fibomax(M)
2 %FIBONACCI: Fibonacci sequence
3 % f = FIBOMAX(M) generates the Fibonacci numbers
4 % until just passing a specified integer M
5 f(1) = 1;
6 f(2) = 2;
7 k = 2;
8 while (f(k) ≤ M)
9     f(k+1) = f(k) + f(k-1);
10    k = k + 1;
11 end
12 j = k - 1; F = f(j);
13 sprintf('Last Fibonacci number before %d is %d ...
14         and occurs after %d months',M,F,j)
14 end
```

## Fibonacci Program

The previous **MatLab** program, `fibomax(4096)` shows it takes 18 months to surpass the population of 4096

Specifically, it shows that the 17<sup>th</sup> Fibonacci number is 2584, while the 18<sup>th</sup> Fibonacci number is 4181

The **Malthusian growth** population model (doubling monthly) gave 4096 after 12 months, thus the Fibonacci delayed growth model increases more slowly

The use of **MatLab** `while` loops is valuable for stopping when a condition is met

## Fibonacci and Golden Ratio

The **Golden ratio**  $\phi = \frac{1+\sqrt{5}}{2} = 1.618033988749895$

Interestingly, the ratio of successive **Fibonacci numbers** in the limit tends to  $\phi$

The program above in MatLab computes the ratio of the first 39 ratios:

```
n = 40; f = fibonacci(n);  
f(2:n) ./ f(1:n-1)
```

The ratios at  $n = 5, 10, 20,$  and  $38$  are

$$\frac{f(5)}{f(4)} = 1.6$$

$$\frac{f(10)}{f(9)} = 1.618181818181818$$

$$\frac{f(20)}{f(19)} = 1.618033998521803$$

$$\frac{f(38)}{f(37)} = 1.618033988749895$$

The last ratio agrees with the **Golden ratio** to double precision



## Collatz Problem

The **Collatz Problem** ([hyperlink](#)) is an unsolved problem in **Number Theory**

The problem is an easily stated sequence of events

- If  $n = 1$ , stop.
- If  $n$  is even, replace it with  $n/2$ .
- If  $n$  is odd, replace it with  $3n + 1$ .

Depending on the choice of  $n$ , it is conjectured that this sequence always terminates after a finite number of steps

We won't try to prove this conjecture, but write a program to follow the sequence for any  $n$

It happens that roundoff error creates problems if an element in the sequence exceeds  $2^{53}$

## $3n + 1$ Sequence

This problem has an iterative process with conditional statements for deciding what to do

This is readily programmed using **MatLab** `if` and `while` statements (See program below)

```
1 function y = f_3nplus1(n)
2 % ``Three n plus 1''.
3 % Study the 3n+1 sequence.
4 y = n;
5 while n > 1
6     if rem(n,2)==0 % Could use mod(n,2)==0
7         n = n/2; % Even numbers/2
8     else
9         n = 3*n+1; % Odd numbers*3 + 1
10    end
11    y = [y n];
12 end
13 end
```

## $3n + 1$ Sequence

Implementation of the previous program produces a sequence ending in 1

`f_3nplus1(7)` yields:

7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

`f_3nplus1(17)` yields:

17 52 26 13 40 20 10 5 16 8 4 2 1

This program quickly computes the sequence

Other programs could be written to study large sets of numbers to try to see patterns

The Lecture Note page provides a GUI program from the text for more elaborate [MatLab](#) code of this problem

# The Number $e$

The Maclaurin series gives  $e = \sum_{j=0}^{\infty} \frac{1}{j!}$

```
1 function eapprox = ecomp(N)
2 %Approximation to e with N terms
3 eapprox = 1;
4 term = 1; tic
5 for jj = 1:N
6     term = term./jj;
7     eapprox = eapprox + term;
8 end
9 toc
10 end
```

Executing the program

```
ecomp(10)
```

yields  $e \approx 2.718281801146385$ , while  $e = 2.718281828459046$

## The Number $e$

**MatLab** has a compact way of writing sums (adding components of a vector)

```
eapprox = sum(1./factorial(0:N))
```

- Computers cannot perform infinite sums, so we must terminate a program
- The program above uses a `for` loop, and we select the number of terms
- Numerically, we usually want a certain accuracy
- This implies setting a tolerance and running a loop until the error in computation is below a set tolerance
- Below is a program for computing  $e$  by truncating the series at a particular tolerance at the last term

## The Number $e$

```
1 function eapprox = esum(tol)
2 %Approximation to e to a given tolerance
3 eapprox = 1;
4 term = eapprox;
5 n = 1;
6 while (abs(term) >= tol)
7     term = term/n;           % New term from old/n
8     eapprox = eapprox + term; % Sum adds new term
9     n = n + 1;
10 end
11 end
```

Executing the program

`esum(1e-4)`

yields  $e \approx 2.718278769841270$ , while  $e = 2.718281828459046$

## Exponential Function, $e^x$

The **Maclaurin series** of  $e^x$  satisfies  $e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$

- Can we write a program for an  $x$  that gives  $e^x$  to a certain tolerance?
- The **ratio test** gives convergence of this series for all  $x$

$$\frac{|x|^{k+1}}{(k+1)!} \cdot \frac{k!}{|x|^k} = \frac{|x|}{k+1} < 1$$

- For any fixed  $x$ , then  $k$  can be taken large enough that the **ratio test** is satisfied
- After a large enough  $k$ , then for the given  $x$  the terms of the series monotonically decrease (in absolute value) toward zero
- Like the previous program, once one term is below the tolerance, then all subsequent terms would be below the tolerance

## Exponential Function, $e^x$

```
1 function esumx = exp_sum(x,tol)
2 %Approximation to e^x to a given tolerance
3 esumx = 1;
4 term = 1;
5 k = 1;
6 while (max(abs(term)) >= tol)
7     term = term.*x./k;    % Recursive formula
8     esumx = esumx + term; % Sum adds new term
9     k = k + 1;
10 end
11 end
```

Executing the program

```
exp_sum([-1 0 1 2 4],1e-4)
```

yields

```
0.3678794 1.0000000 2.7182818 7.3890561 54.5981500
```





## Exponential Function, $e^x$

Important Points in the program:

- **Recursive formula**, where *new term* builds off of *old term*
- Series expansion:

$$e^x = 1 + 1 \cdot x + x \cdot \frac{x}{2} + \frac{x^2}{2!} \cdot \frac{x}{3} + \frac{x^3}{3!} \cdot \frac{x}{4} + \dots$$

- Each iteration through loop adds the *new term* to the *sum*
- The *new terms* are compared the user supplied *tolerance*
- Provided the *new term* is greater than the *tolerance* the sum is increased, otherwise the program terminates
- This program is set up for **vector**  $x$ , so the new terms and the sums are vectors
- Want all *new terms* less than the *tolerance* before exiting program

## Graph of Exponential Function, $e^x$

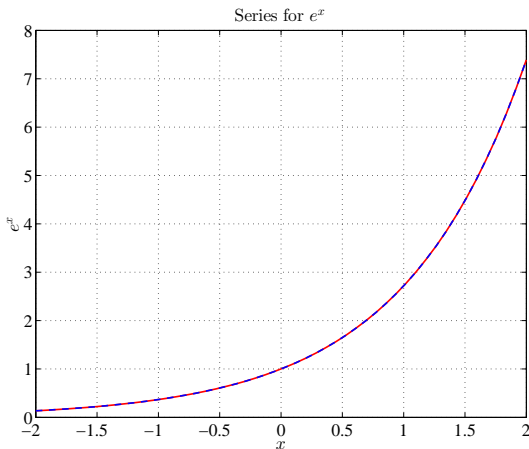
```
1 function exp_plot(Lx,res,tol)
2 % Create a plot using the exp_sum function
3 tic
4 xx = linspace(-Lx,Lx,res); % x points of evaluation
5 yy = exp_sum(xx,tol); % run exp series program
6 yy1 = exp(xx); % evaluate  $e^x$  with MatLab
7
8 figure(101) % assign a figure number
9
10 plot(xx,yy,'r-','LineWidth',1.5); % plot the series
11 hold on % add more graphs
12 plot(xx,yy1,'b--','LineWidth',1.5); % plot  $e^x$ 
13 grid; % provide gridlines
```

## Graph of Exponential Function, $e^x$

```
15 % Set up fonts and labels for the Graph
16 fontlabs = 'Times New Roman';
17 xlabel('$x$', 'FontSize', 16, 'FontName', fontlabs, ...
18     'interpreter', 'latex');
19 ylabel('$e^x$', 'FontSize', 16, 'FontName', fontlabs, ...
20     'interpreter', 'latex');
21 mytitle = 'Series for $e^x$';
22 title(mytitle, 'FontSize', 16, 'FontName', ...
23     'Times New Roman', 'interpreter', 'latex');
24 set(gca, 'FontSize', 16);
25
26 %print -depsc exp_gr.eps           % Create EPS file ...
    (Figure)
27 end
```

## Graph of Exponential Function, $e^x$

Result of the previous program with  $Lx = 2$ ,  $res = 100$ , and  $tol = 1e - 3$  shown below:



# Bessel Functions

**Bessel's Equation** is an important differential equation:

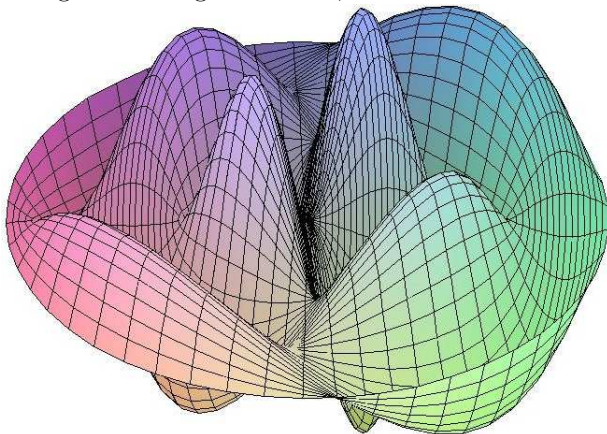
$$x^2 \frac{d^2 y}{dx^2} + x \frac{dy}{dx} + (x^2 - m^2)y = 0$$

- This equation appears naturally in applied problems with circular or cylindrical geometry - vibrating drums, aircraft design, ...
- Equation is a regular singular differential equation
- Solved using **Method of Frobenius** - A power series technique
- Series solution,  $m^{\text{th}}$  order Bessel function of  $1^{\text{st}}$  kind ( $m$  an integer) is given by:

$$J_m(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{n!(m+n)!} \left(\frac{x}{2}\right)^{2n+m}$$

## Bessel Function, $J_m(x)$

Figure showing a vibrating membrane, which uses a **Bessel function**



# $m^{\text{th}}$ Order Bessel Function

The  $m^{\text{th}}$  **Order Bessel Function of 1<sup>st</sup> Kind** satisfies:

$$J_m(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{n!(m+n)!} \left(\frac{x}{2}\right)^{2n+m}$$

- Must find a **recursive relationship**
- Create appropriate stopping criteria with `while` loop based on a **tolerance**
- Include a condition to avoid too many steps
- **Ratio Test** shows this series converges for all  $x$

$m^{\text{th}}$  Order Bessel FunctionFind **recursive relation**:

$$\begin{aligned}
 J_m(x) &= \sum_{n=0}^{\infty} \frac{(-1)^n}{n!(m+n)!} \left(\frac{x}{2}\right)^{2n+m} \\
 &= \frac{1}{m!} \left(\frac{x}{2}\right)^m - \frac{1}{1!(m+1)!} \left(\frac{x}{2}\right)^{(m+2)} + \frac{1}{2!(m+2)!} \left(\frac{x}{2}\right)^{(m+4)} - \dots \\
 &= \frac{1}{m!} \left(\frac{x}{2}\right)^m + \frac{1}{m!} \left(\frac{x}{2}\right)^m \cdot \frac{(-1)}{(m+1)} \left(\frac{x}{2}\right)^2 \\
 &\quad + \frac{1}{1!(m+1)!} \left(\frac{x}{2}\right)^{(m+2)} \cdot \frac{1}{2(m+2)} \left(\frac{x}{2}\right)^2 + \dots
 \end{aligned}$$

Thus,

$$\text{New Term} = \text{Old Term} \cdot \left(-\frac{x}{2}\right)^2 / ((n+1) \cdot (n+m+1))$$



# Bessel Function, $J_m(x)$

Below is the **MatLab** code for  $J_m(x)$

```
1 function tot = bessell(x,m,tol)
2 % Approximation to  $J_m(x)$  to a given tolerance
3 tot = (x/2).^m/factorial(m);
4 term = tot;
5 k = 1;
6 while (max(abs(term)) >= tol)
7     term = term.*(-(x/2).^2/(k*(k+m)));
8     tot = tot + term;
9     k = k + 1;
10 end
11 end
```

# Bessel Function, $J_m(x)$

Below is the **MatLab** code for  $J_m(x)$  with a maximum number of steps

```
1 function tot = bessel2(x,m,tol,Nmax)
2 % Approximation to J_m(x) to a given tolerance
3 % or Nmax terms
4 tot = (x/2).^m/factorial(m);
5 term = tot;
6 k = 1;
7 while ((max(abs(term)) >= tol)&& (k <= Nmax))
8     term = term.*(-(x/2).^2/(k*(k+m)));
9     tot = tot + term;
10    k = k + 1;
11 end
12 end
```

Graph of  $J_m(x)$ 

Next 2 slides show Bessel plot routine, then a graph

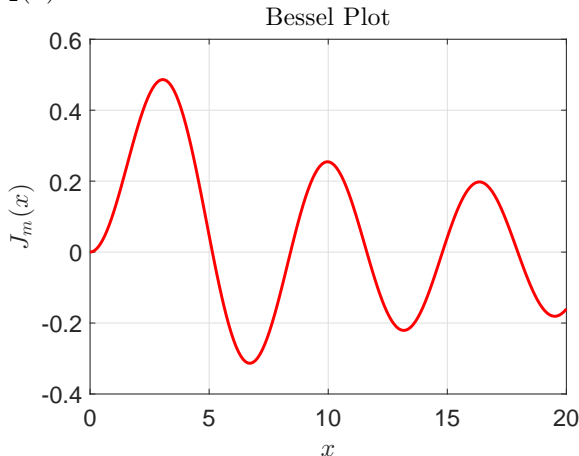
```
1 function bessel_plot(m,Lx,res,tol)
2 % Create a plot using the bessel function
3
4 xx = linspace(0,Lx,res); % x points of evaluation
5 yy = bessel(xx,m,tol); % run bessel series program
6
7 figure(101) % assign a figure number
8
9 plot(xx,yy,'r-','LineWidth',2.0); % plot the series
10 grid; % provide gridlines
```

Graph of  $J_m(x)$ 

```
12 % Set up fonts and labels for the Graph
13 fontlabs = 'Times New Roman';
14 xlabel('$x$', 'FontSize', 16, 'FontName', fontlabs, ...
15     'interpreter', 'latex');
16 ylabel('$J_m(x)$', 'FontSize', 16, 'FontName', fontlabs, ...
17     'interpreter', 'latex');
18 mytitle = 'Bessel Plot';
19 title(mytitle, 'FontSize', 16, 'FontName', ...
20     'Times New Roman', 'interpreter', 'latex');
21 set(gca, 'FontSize', 16);
22
23 print -depsc bessel-gr.eps      % Create EPS file ...
    (Figure)
24 end
```

## Graph of $J_2(x)$

Graph of  $J_2(x)$



## Graph of Multiple Bessel Functions

```
1 function bessel2_plot(Lx,res,tol)
2 % Create a plot using the bessel function
3
4 xx = linspace(0,Lx,res); % x points of evaluation
5 yy = bessel(xx,0,tol); % run bessel series program
6
7 figure(102) % assign a figure number
8
9 plot(xx,yy,'k-','LineWidth',2.0); % plot the series
10 hold on
11 grid; % provide gridlines
```

## Graph of Multiple Bessel Functions

```
12 for m = 1:3
13     if (m == 1)
14         lsty = 'b-';
15     elseif (m == 2)
16         lsty = 'r-';
17     else
18         lsty = 'm-';
19     end
20     yy = bessel(xx,m,tol);
21     plot(xx,yy,lsty,'LineWidth',2.0);
22 end
23 h = legend('$J_0(x)$', '$J_1(x)$', '$J_2(x)$', ...
24           '$J_3(x)$', ...
25           'Location','northeast');
26 set(h,'Interpreter','latex')
```

## Graph of Multiple Bessel Functions

```
27 % Set up fonts and labels for the Graph
28 fontlabs = 'Times New Roman';
29 xlabel('$x$', 'FontSize', 16, 'FontName', fontlabs, ...
30     'interpreter', 'latex');
31 ylabel('$J_m(x)$', 'FontSize', 16, 'FontName', fontlabs, ...
32     'interpreter', 'latex');
33 mytitle = 'Bessel Plot';
34 title(mytitle, 'FontSize', 16, 'FontName', ...
35     'Times New Roman', 'interpreter', 'latex');
36 set(gca, 'FontSize', 16);
37
38 print -depsc bessel2_gr.eps    % Create EPS file ...
39                               (Figure)
39 end
```



## Graph of Multiple Bessel Functions

Graph of  $J_0(x)$ ,  $J_1(x)$ ,  $J_2(x)$ , and  $J_3(x)$

Bessel Plot

