A few more comments about MATLAB.

**Functions as Arguments**

```matlab
function yprime = diff (f, x, h);
yprime = (feval(f,x+h/2)-feval(f,x-h/2))/h;
```

Notice that MATLAB functions just end at the end of an m-file. This would be in a file called diff.m.

Calling line from a prompt or m-file

```matlab
>> yprime = diff(@sin,pi/8,1e-4)
gives an approximation to the derivative of sin \(x\) at \(x = \pi/8\).
```

Functions can be defined in line

```matlab
>> fun = inline(’1/(x+1)’);
```

Then the call may be used

```matlab
>> fun_hand = @(fun
>> yprime = diff(fun_hand,0,1e-4);
```

Function handles are also defined for functions you define.

On problem 2, p. 7. I did not talk about binary arithmetic specifically. That problem asks you to write down representations of

\[ \frac{1}{3}, \frac{1}{5}, \frac{1}{6}, \frac{1}{7}, \frac{1}{9}, \frac{1}{10}. \]

For example,

\[ \frac{1}{3} = d_1 \cdot \frac{1}{2} + d_2 \cdot \frac{1}{4} + d_3 \cdot \frac{1}{8} + \ldots \]

and \(d_i \in \{0, 1\}\). Probably the easiest way is to use long division, taking \((11)_2\) into \(1.000\ldots\). The solution is

\[ \frac{1}{3} = 0.0\bar{1}. \]

Or in the way the book wants you to answer the question

\[ \frac{1}{3} = 0.10\bar{1} \times 2^{-1} = 1.0\bar{1} \times 2^{-2}. \]

**Chapter Seven – Linear Systems of Equations**

\[ Ax = b \]

\[ 1 \]
where
\[
A = \begin{pmatrix}
  a_{11} & a_{12} & \cdots & a_{1n} \\
  a_{21} & a_{22} & \cdots & a_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  a_{n-1,1} & a_{n-1,2} & \cdots & a_{n-1,n} \\
  a_{n1} & a_{n2} & \cdots & a_{nn}
\end{pmatrix}
\]
\[
\mathbf{x} = \begin{pmatrix}
x_1 \\
x_2 \\
\vdots \\
x_n
\end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix}
b_1 \\
b_2 \\
\vdots \\
b_n
\end{pmatrix}.
\]

We write this in full blown form as
\[
\begin{align*}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\
\vdots \\
a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n
\end{align*}
\]

Solving \(Ax = b\) in MATLAB is a “one-liner.”
\[
x = A\backslash b
\]

We are going to look at what is inside of this statement!

We now briefly consider the example
\[
\begin{align*}
2x_1 + 3x_2 &= 8 \\
x_1 - 5x_2 &= -9
\end{align*}
\]

Using the augmented matrix approach we have
\[
\begin{pmatrix}
  2 & 3 & 8 \\
  1 & -5 & -9
\end{pmatrix}.
\]

The first thing we do is multiply 0.5 times the first equation and subtract it from the second to get
\[
\begin{pmatrix}
  2 & 3 & 8 \\
  0 & -6.5 & -13
\end{pmatrix}.
\]

That becomes the system
\[
\begin{align*}
2x_1 + 3x_2 &= 8 \\
-6.5 \ x_2 &= -13
\end{align*}
\]
Back substitution yields

\[ x_2 = -13/(-6.5) = 2, \quad x_1 = (8 - 3x_2)/2 = 1. \]

The last system is

\[ Ux = y, \]

where

\[ U = \begin{pmatrix} 2 & 3 \\ 0 & -6.5 \end{pmatrix}, \quad y = \begin{pmatrix} 8 \\ -13 \end{pmatrix}. \]

We note that

\[ A = LU, \quad b = Ly, \]

where

\[ L = \begin{pmatrix} 1 & 0 \\ 0.5 & 1 \end{pmatrix}. \]

Notice the strategic position of 0.5, the factor used in elimination. Also note that \( L \) and \( U \) may be computed without reference to \( b \).

So a procedure to solve this problem is

\[
\begin{align*}
A &= LU, & \text{Factorization step} \\
Ly &= b, & \text{Forward substitution} \\
Ux &= y, & \text{Back substitution}
\end{align*}
\]

The first line is called the LU factorization of \( A \). Gaussian elimination is an algorithm to compute that factorization. The remaining two steps can be used for more than one right hand side by repeating them over and over again.

For instance, let

\[ B = (b_1, \ldots, b_p), \quad X = (x_1, \ldots, x_p) \]

and suppose that

\[ Ax_j = b_j, \quad j = 1, \ldots, p. \]

Then we do one LU factorization and solve

\[
\begin{align*}
Ly_j &= b_j, & j &= 1, \ldots, p \\
Ux_j &= y_j
\end{align*}
\]
Better is probably to solve

\[ LY = B, \quad Y = (y_1, \ldots, y_p) \]
\[ UX = Y \]

**Matrix and Vector Operations**

Three “levels” based upon the development of the BLAS.

**Level One–Vector Operations**

\[ y \leftarrow y + \alpha x, \quad \text{AXPY} \]

In componentwise code this is

```plaintext
for i = 1: n
    y(i) = y(i) + alpha*x(i);
end;
```

A second such operation is dot product.

\[ \text{dot} = x^T y = \sum_{j=1}^{n} x_j y_i. \]

All of these are \( O(n) \) operations with \( O(n) \) data.
We will introduce other such operations later.

**Level Two–Matrix-Vector Operations**

\[ y \leftarrow y + Ax, \quad \text{GAXPY} \]

These are \( O(n^2) \) operations with \( O(n^2) \) data.

In MATLAB a columnwise version of this is a sequence of AXPY’s.

```plaintext
for j=1: n
    y = y + x(j) * A(:, j);
end;
```

A rowwise version is a sequence of dot products.
for i=1:m
    y(i) = y(i) + A(i,:) * x;
end;

Again other possibilities may be suggested later.

Level Three– Matrix Operations

\[ C = C + AB, \quad C \in \mathbb{R}^{m \times p}, A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{n \times p}. \]

You were probably taught that matrix multiplication is something like this.

for \( j = 1: p \)
  for \( i = 1: m \)
    for \( k = 1: n \)
      \[ C(i,j) = C(i,j) + A(i,k) * B(k,j); \]
    end;
  end;
end;

It might be better to break it down into a bunch of GAXPYs. That is

for \( j = 1: p \)
  \[ C(:,j) = C(:,j) + A * B(:,j); \]
end;

Level three operations require \( O(n^3) \) operations with \( O(n^2) \) data.