PETUNJUK PRAKTIKUM

PRAKTIKUM

PENGOLAHAN SINYAL DIGITAL

Laboratorium Dasar
Teknik Elektro

Sekolah Teknik Elektro Dan Informatika
Institut Teknologi Bandung

2022

BUKU PETUNJUK
PRAKTIKUM PENGOLAHAN SISTEM DIGITAL
EL 3110

Mervin T. Hutabarat

Armein Z. R. Langi

Yoanes Bandung

Erwin Cahyadi

Nina Lestari

Egi M.I. Hidayat

Yulyan Wahyu Hadi

Kelvin Sutirta

Laboratorium Dasar Teknik Elektro

Sekolah Teknik Elektro Dan Informatika

Institut Teknologi Bandung

2022

DAFTAR ISI

ATURAN UMUM LABORATORIUM

KELENGKAPAN

Setiap praktikan wajib berpakaian lengkap, mengenakan celana panjang/ rok, kemeja dan mengenakan sepatu. Praktikan wajib membawa kelengkapan berikut:

- Modul praktikum
- Buku Catatan Laboratorium (BCL)
- Alat tulis (dan kalkulator, jika diperlukan)
- *Name tag*
- Kartu Praktikum

PERSIAPAN

SEBELUM PRAKTIKUM

- Membaca dan memahami isi modul praktikum
- Mengerjakan hal-hal yang dapat dikerjakan sebelum praktikum dilaksanakan, misalnya mengerjakan soal perhitungan, membuat *source code*, mengisi Kartu Praktikum dll.
- Mengerjakan Tugas Pendahuluan
- Mengisi daftar hadir
- Mengambil kunci loker dan melengkapi administrasi peminjaman kunci loker (tukarkan dengan kartu identitas: KTM/ SIM/ KTP)

SELAMA PRAKTIKUM

- Perhatikan dan kerjakan setiap percobaan dengan waktu sebaik-baiknya, diawali dengan kehadiran praktikan secara tepat waktu
- Kumpulkan Kartu Praktikum pada asisten
- Dokumentasikan pada BCL (lihat Petunjuk Penggunaan BCL) tentang hal-hal penting terkait percobaan yang sedang dilakukan

SETELAH PRAKTIKUM

- Memastikan BCL telah ditandatangani oleh asisten,
- Mengembalikan kunci loker dan melengkapi administrasi pengembalian kunci loker (pastikan kartu identitas KTM/ SIM/ KTP diperoleh kembali),

- Mengerjakan laporan dalam bentuk SoftCopy (lihat Panduan Penyusunan Laporan di laman http://ldte.stei.itb.ac.id)),
- Mengirimkan file dengan cara mengunggah di laman http://praktikum.stei.itb.ac.id.. Waktu pengiriman paling lambat jam 11.00 WIB, dua hari kerja berikutnya setelah praktikum kecuali ada kesepakatan lain antara Dosen Pengajar dan/ atau Asisten

PERGANTIAN JADWAL

KASUS BIASA

- Lihatlah format Pertukaran Jadwal di http://ldte.stei.itb.ac.id pada halaman Panduan
- Salah satu praktikan yang bertukar jadwal harus mengirimkan e-mail ke labdasar@stei.itb.ac.id . Waktu pengiriman paling lambat jam 16.30, sehari sebelum praktikum yang dipertukarkan
- Pertukaran diperbolehkan setelah ada email konfirmasi dari Lab. Dasar

KASUS SAKIT ATAU URUSAN MENDESAK PRIBADI LAINNYA

Jadwal pengganti dapat diberikan kepada praktikan yang sakit atau memiliki urusan mendesak pribadi.

- Praktikan yang hendak mengubah jadwal untuk urusan pribadi mendesak harus memberitahu staf tata usaha laboratorium sebelum jadwal praktikumnya melalui email.
- Segera setelah praktikan memungkinkan mengikuti kegiatan akademik, praktikan dapat mengikuti praktikum pengganti setelah mendapatkan konfirmasi dari staf tata usaha laboratorium dengan melampirkan surat keterangan dokter bagi yang sakit atau surat terkait untuk yang memiliki urusan pribadi.

KASUS ”KEPENTINGAN MASSAL”

- ”Kepentingan massal” terjadi jika ada lebih dari 1/3 rombongan praktikan yang tidak dapat melaksanakan praktikum pada satu hari yang sama karena alasan yang terkait kegiatan akademis

- Isi Form Pergantian Jadwal dan serahkan pada TU Lab. Dasar secepatnya. Jadwal praktikum pengganti satu hari itu akan ditentukan kemudian oleh Kordas praktikum yang bersangkutan

SANKSI

Pengabaian aturan-aturan di atas dapat dikenakan sanksi pengguguran nilai praktikum terkait.

PANDUAN UMUM KESELAMATAN DAN PENGGUNAAN PERALATAN LABORATORIUM

KESELAMATAN

Pada prinsipnya, untuk mewujudkan praktikum yang aman diperlukan partisipasi seluruh praktikan dan asisten pada praktikum yang bersangkutan. Dengan demikian, kepatuhan setiap praktikan terhadap uraian panduan pada bagian ini akan sangat membantu mewujudkan praktikum yang aman.

BAHAYA LISTRIK

- Perhatikan dan pelajari tempat-tempat sumber listrik (stop-kontak dan *circuit breaker*) dan cara menyala-matikannya. Jika melihat ada kerusakan yang berpotensi menimbulkan bahaya, laporkan pada asisten
- Hindari daerah atau benda yang berpotensi menimbulkan bahaya listrik (sengatan listrik/ strum) secara tidak disengaja, misalnya kabel jala-jala yang terkelupas dll.
- Tidak melakukan sesuatu yang dapat menimbulkan bahaya listrik pada diri sendiri atau orang lain
- Keringkan bagian tubuh yang basah karena, misalnya, keringat atau sisa air wudhu
- Selalu waspada terhadap bahaya listrik pada setiap aktivitas praktikum

Kecelakaan akibat bahaya listrik yang sering terjadi adalah tersengat arus listrik. Berikut ini adalah hal-hal yang harus diikuti praktikan jika hal itu terjadi:

- Jangan panik
- Matikan semua peralatan elektronik dan sumber listrik di meja masing-masing dan di meja praktikan yang tersengat arus listrik
- Bantu praktikan yang tersengat arus listrik untuk melepaskan diri dari sumber listrik
- Beritahukan dan minta bantuan asisten, praktikan lain dan orang di sekitar anda tentang terjadinya kecelakaan akibat bahaya listrik

BAHAYA API ATAU PANAS BERLEBIH

- Jangan membawa benda-benda mudah terbakar (korek api, gas dll.) ke dalam ruang praktikum bila tidak disyaratkan dalam modul praktikum
- Jangan melakukan sesuatu yang dapat menimbulkan api, percikan api atau panas yang berlebihan
- Jangan melakukan sesuatu yang dapat menimbulkan bahaya api atau panas berlebih pada diri sendiri atau orang lain
- Selalu waspada terhadap bahaya api atau panas berlebih pada setiap aktivitas praktikum

Berikut ini adalah hal-hal yang harus diikuti praktikan jika menghadapi bahaya api atau panas berlebih:

- Jangan panik
- Beritahukan dan minta bantuan asisten, praktikan lain dan orang di sekitar anda tentang terjadinya bahaya api atau panas berlebih
- Matikan semua peralatan elektronik dan sumber listrik di meja masing-masing
- Menjauh dari ruang praktikum

BAHAYA BENDA TAJAM DAN LOGAM

- Dilarang membawa benda tajam (pisau, gunting dan sejenisnya) ke ruang praktikum bila tidak diperlukan untuk pelaksanaan percobaan
- Dilarang memakai perhiasan dari logam misalnya cincin, kalung, gelang dll.
- Hindari daerah, benda atau logam yang memiliki bagian tajam dan dapat melukai
- Tidak melakukan sesuatu yang dapat menimbulkan luka pada diri sendiri atau orang lain

LAIN-LAIN

- Dilarang membawa makanan dan minuman ke dalam ruang praktikum

PENGGUNAAN PERALATAN PRAKTIKUM

Berikut ini adalah panduan yang harus dipatuhi ketika menggunakan alat-alat praktikum:

- Sebelum menggunakan alat-alat praktikum, pahami petunjuk penggunaan alat itu. Petunjuk penggunaan beberapa alat dapat didownload di http://labdasar.ee.itb.ac.id
- Perhatikan dan patuhi peringatan (*warning*) yang biasa tertera pada badan alat

- Pahami fungsi atau peruntukan alat-alat praktikum dan gunakanlah alat-alat tersebut hanya untuk aktivitas yang sesuai fungsi atau peruntukannya. Menggunakan alat praktikum di luar fungsi atau peruntukannya dapat menimbulkan kerusakan pada alat tersebut dan bahaya keselamatan praktikan
- Pahami *rating* dan jangkauan kerja alat-alat praktikum dan gunakanlah alat-alat tersebut sesuai *rating* dan jangkauan kerjanya. Menggunakan alat praktikum di luar *rating* dan jangkauan kerjanya dapat menimbulkan kerusakan pada alat tersebut dan bahaya keselamatan praktikan
- Pastikan seluruh peralatan praktikum yang digunakan aman dari benda/ logam tajam, api/ panas berlebih atau lainnya yang dapat mengakibatkan kerusakan pada alat tersebut
- Tidak melakukan aktifitas yang dapat menyebabkan kotor, coretan, goresan atau sejenisnya pada badan alat-alat praktikum yang digunakan

SANKSI

Pengabaian uraian panduan di atas dapat dikenakan sanksi tidak lulus mata kuliah praktikum yang bersangkutan

TABEL SANKSI PRAKTIKUM

Berlaku mulai: 14 Agustus 2017

Level	Waktu	Kasus	Sanksi	Pengurangan nilai per modul
Akademik	Saat dan setelah praktikum	Semua kegiatan plagiasi (mencontek): tugas pendahuluan, test dalam praktikum, laporan praktikum	Gugur praktikum	
		Sengaja tidak mengikuti praktikum		
Berat	Saat praktikum	Tidak hadir praktikum	Gugur modul	
		Terlambat hadir praktikum		
		Pakaian tidak sesuai: kemeja, sepatu		
		Tugas pendahuluan tidak dikerjakan/hilang/tertinggal		
Ringan	Saat Praktikum	Pertukaran jadwal tidak sesuai aturan/ketentuan		-25 nilai akhir
		Tidak mempelajari modul sebelum praktikum/tidak mengerti isi modul	Dikeluarkan dari praktikum	-25 nilai akhir
		BCL tertinggal/hilang		-100% nilai BCL
		Name Tag tertinggal/hilang		-10 nilai akhir
		Kartu praktikum tertinggal/hilang		-25 nilai akhir
		Kartu praktikum tidak lengkap data dan foto		-10 nilai akhir
		Loker tidak dikunci/kunci tertinggal		-10 nilai akhir
	Setelah Praktikum	Tidak ada paraf asisten di BCL/kartu praktikum		-25 nilai akhir
		Terlambat mengumpulkan laporan		-1/min nilai akhir, maks -50
		Terlambat mengumpulkan BCL		-1/min nilai BCL, maks -50
		Tidak bawa kartu praktikum saat pengumpulan BCL		-50 nilai BCL
		Tidak minta paraf admin saat pengumpulan BCL		-50 nilai BCL

Catatan:

1. Pelanggaran akademik menyebabkan gugur praktikum, nilai praktikum E
2. Dalam satu praktikum, praktikan maksimal boleh melakukan
    a. 1 pelanggaran berat dan 1 pelanggaran ringan; atau
    b. 3 pelanggaran ringan
3. Jika jumlah pelanggaran melewati point 2, praktikan dianggap gugur praktikum.
4. Praktikan yang terkena sanksi gugur modul wajib mengganti praktikum pada hari lain dengan nilai modul tetap 0. Waktu pengganti praktikum ditetapkan bersama asisten. Jika praktikan tidak mengikuti ketentuan praktikum (pengganti) dengan baik, akan dikenakan sanksi gugur praktikum.
5. Setiap pelanggaran berat dan ringan dicatat/diberikan tanda di kartu praktikum
6. Waktu acuan adalah waktu sinkron dengan NIST
7. Sanksi yang tercantum di tabel adalah sanksi minimum.
8. Sanksi yang belum tercantum akan ditentukan kemudian.

PERCOBAAN I

PENGENALAN MATLAB

MATLAB akan digunakan secara ekstensif pada praktikum ini. Modul ini memberikan tinjauan singkat mengenai MATLAB dan kapabilitasnya dengan penekanan pada isu pemrograman. Seluruh materi pada modul ini merupakan terjemahan bebas dari buku "*DSP First, A Multimedia Approach*" karangan James H. McClellan, Ronald W. Schafer, dan Mark A. Yoder, terbitan Prentice-Hall (1998), khususnya Appendix B dan C.

1. TUJUAN

1. Mempelajari penggunaan sistem help untuk mengetahui *commands* dan *syntax* dasar MATLAB
2. Dapat menggunakan MATLAB untuk desain filter
3. Mempelajari bagaimana menulis fungsi dan *m-file* pada MATLAB
4. Merancang pem-*filter*-an FIR dengan MATLAB
5. Memahami pem-*filter*-an lewat MATLAB secara mendalam

2. DASAR TEORI

MATLAB (Matrix Laboratory) adalah sebuah program untuk analisis dan komputasi numerik. Pada awalnya, program ini merupakan *interface* untuk koleksi rutin-rutin numerik dari proyek LINPACK dan EISPACK, namun sekarang merupakan produk komersial dari perusahaan Mathworks, Inc. MATLAB telah berkembang menjadi sebuah *environment* pemrograman yang canggih yang berisi fungsi-fungsi *built-in* untuk melakukan tugas pengolahan sinyal, aljabar linier, dan kalkulasi matematis lainnya. MATLAB juga berisi *toolbox* yang berisi fungsi-fungsi tambahan untuk aplikasi khusus .

MATLAB bersifat *extensible*, dalam arti bahwa seorang pengguna dapat menulis fungsi baru untuk ditambahkan pada *library* ketika fungsi-fungsi *built-in* yang tersedia tidak dapat melakukan tugas tertentu. Kemampuan pemrograman yang dibutuhkan tidak terlalu sulit bila Anda telah memiliki pengalaman dalam pemrograman bahasa lain seperti C, PASCAL, atau FORTRAN.

2.1. MATLAB HELP

MATLAB menyediakan sistem *help on-line* yang dapat diakses dengan perintah help. Misalnya, untuk memperoleh informasi mengenai fungsi filter, Anda hanya perlu mengetikkan perintah

```
>> help filter
```

Perintah di atas akan menampilkan informasi dalam bentuk teks pada layar MATLAB Anda. Sebuah perintah yang sangat berguna untuk mempelajari pemrograman MATLAB adalah intro, yang membahas konsep-konsep dasar tentang bahasa MATLAB. Selain itu, juga terdapat banyak program demonstrasi yang mengilustrasikan berbagai kapabilitas MATLAB, yang dapat dimulai dengan perintah demo.

2.2. VARIABEL DAN OPERASI MATRIKS

Tipe variabel dasar pada MATLAB adalah matriks (pada versi 5 dan ke atas, MATLAB juga menyediakan berbagai tipe data seperti pada bahasa pemrograman lainnya). Untuk mendeklarasikan sebuah variabel, Anda hanya perlu memberikan nilai tertentu padanya pada MATLAB *prompt*. Sebagai contoh,

```
>> M = [ 1   2   6; 5   2   1]

   M =

        1      2      6

        5      2      1
```

Ketika definisi sebuah matriks melibatkan sebuah rumus yang panjang atau banyak entri, maka sebuah perintah MATLAB yang sangat panjang dapat dipecah menjadi dua (atau lebih) baris dengan cara menempatkan sebuah tanda (…) pada akhir dari sebuah baris yang ingin dilanjutkan. Sebagai contoh,

```
P = [ 1, 2, 4, 6, 8 ]+ [ pi, 4, exp(1), 0, -1] + …

      [ cos(0.1*pi), sin(pi/3), tan(3), atan(2), sqrt(pi) ];
```

Ketika sebuah ekspresi perintah atau pernyataan diakhiri dengan tanda *semicolon* (;), maka hasilnya tidak akan ditampilkan di layar. Hal ini sangat membantu ketika Anda bekerja dengan matriks dengan ukuran yang sangat besar.

Ukuran dari sebuah matriks dapat diketahui dengan operator size:

```
>> Msize  = size(M)

    Msize  =

             2      3
```

Oleh karena itu, kita tidak perlu menggunakan variabel khusus untuk melacak jumlah baris dan kolom suatu matriks. Ada dua jenis variabel matriks pada MATLAB, yakni skalar (*scalars*) dan vektor (*vectors*). Sebuah skalar adalah sebuah matriks yang hanya berisi satu elemen, jadi berukuran 1 x 1. Sebuah vektor adalah sebuah matriks yang hanya berisi satu baris atau kolom.

Elemen individu dari sebuah variabel matriks dapat diakses dengan memberikan indeks baris dan kolom, sebagai contoh

```
>> M13  = M(1,3)

     M13 =

                6
```

Submatriks juga dapat diakses dengan cara yang mirip dengan menggunakan operator *colon* (:) seperti yang dijelaskan pada sesi berikut.

2.2. 1. OPERATOR COLON (:)

Operator *colon* (:) sangat berguna untuk membuat *index arrays*. Gunakan perintah help colon untuk mengetahui deskripsi detail tentang kapabilitasnya.

Notasi *colon* didasarkan pada ide bahwa sebuah selang indeks dapat dihasilkan dengan memberikan sebuah nilai awal, interval, dan sebuah nilai akhir. Karena itu, sebuah vektor yang terpartisi secara teratur dapat diperoleh dengan perintah

iii = nilai awal : interval : nilai akhir

Tanpa parameter interval, nilai *default*-nya adalah 1. Metode perhitungan ini mirip dengan notasi *loop* DO pada FORTRAN, namun metode pada MATLAB selangkah lebih maju dengan cara menggabungkannya dengan pengindeksan matriks. Untuk sebuah matriks A 9 x 8, A(2,3) adalah elemen skalar yang berada pada baris kedua dan kolom ketiga dari matriks A. Jadi sebuah submatriks 4 x 3 dapat diekstrak dengan perintah A(2:5,1:3). Tanda colon juga berfungsi sebagai sebuah *wild card*, misalnya, A(2,:) adalah baris kedua matriks A.

Pengindeksan mundur akan membalikkan sebuah vektor, misalnya X(9:-1:1) untuk sebuah vektor yang berisi 9 buah elemen. Kadang-kadang, Anda juga memerlukan sebuah daftar yang berisi semua nilai elemen pada matriks, jadi A(:) memberikan sebuah vektor kolom 72 x 1, yang merupakan hasil *concatenation* elemen-elemen setiap kolom matrik A. Ini merupakan contoh *reshaping* matriks. Teknik *reshaping* yang lebih umum dapat dilakukan dengan fungsi reshape(A,M,N). Sebagai contoh, matriks A 9 x 8 dapat di-*reshape* menjadi sebuah matriks 12 x 6 dengan Anew = reshape(A,12,6).

2.2.2. OPERASI MATRIKS DAN ARRAY

Operasi *default* pada MATLAB adalah operasi matriks. Jadi A*B berarti perkalian matriks, yang akan dibahas pada bagian berikut.

2.2.2a. Tinjauan Perkalian Matriks

Operasi perkalian matriks AB hanya dapat dilakukan bila kedua matriks tersebut memiliki dimensi yang kompatibel, yakni jumlah kolom matriks A harus sama dengan jumlah baris matriks B. Sebagai contoh, sebuah matriks 5 x 8 dapat mengalikan sebuah matriks 8 x 3 untuk menghasilkan sebuah matriks AB 5 x 3. Secara umum, bila A adalah m x n, maka B haruslah n x p, dan hasil perkalian AB akan memiliki dimensi m x p. Umumnya perkalian matriks tidak bersifat komutatif, yakni $AB \neq BA$. Bila $p \neq m$, maka perkalian AB tidak terdefinisi.

Beberapa kasus khusus untuk perkalian matriks adalah *outer product* dan *inner product*. Pada *outer product*, sebuah vektor kolom mengalikan sebuah vektor baris untuk menghasilkan sebuah matriks. Bila kita membiarkan semua elemen salah satu vektor tersebut berupa '1' , maka kita akan memperoleh hasil yang berulang.

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} a_1 & a_1 & a_1 & a_1 \\ a_2 & a_2 & a_2 & a_2 \\ a_3 & a_3 & a_3 & a_3 \end{bmatrix}$$

Untuk *inner product*, sebuah vektor baris mengalikan sebuah vektor kolom, jadi hasilnya berupa skalar. Bila kita membiarkan semua elemen salah satu vektor tersebut berupa '1', maka kita akan memperoleh penjumlahan semua elemen vektor lainnya.

$$\begin{bmatrix} a_1 & a_2 & a_3 & a_4 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = a_1 + a_2 + a_3 + a_4$$

2.2.2b. Operasi pointwise array

Bila kita ingin melakukan perkalian *pointwise*, ada beberapa kebingungan yang bisa muncul. Pada kasus *pointwise*, kita ingin mengalikan matriks secara elemen per elemen, jadi mereka harus memiliki dimensi yang sama. Sebagai contoh, dua matriks 5 x 8 dapat dikalikan secara *pointwise*, walaupun keduanya tidak bisa melakukan perkalian matriks biasa. Untuk melakukan perkalian *pointwise* pada MATLAB, kita menggunakan operator "*point-star"*A .* B. Misalnya bila A dan B keduanya adalah matriks 3 x 2 maka

$$C = A.*B = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} .* \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} = \begin{bmatrix} a_{11} * b_{11} & a_{12} * b_{12} \\ a_{21} * b_{21} & a_{22} * b_{22} \\ a_{31} * b_{31} & a_{32} * b_{32} \end{bmatrix}$$

Untuk selanjutnya, perkalian semacam ini kita sebut dengan istilah perkalian *array*. Perhatikan bahwa perkalian *array* bersifat komutatif karena kita akan memperoleh hasil yang sama bila kita menghitung D = B.*A.

Dalam MATLAB, bila sebuah "titik" digunakan dengan operator aritmetik, maka ia akan mengubah definisi operator tersebut ke operasi *pointwise*. Jadi operator ./ berarti pembagian *pointwise*, .^ berarti pemangkatan *pointwise*. Misalnya, xx = (0.9).^(0:49) akan menghasilkan suatu vector yang nilainya sama dengan $(0,9)^n$ untuk n = 0,1, 2, …49.

2.2.2c. Operasi concatenation array

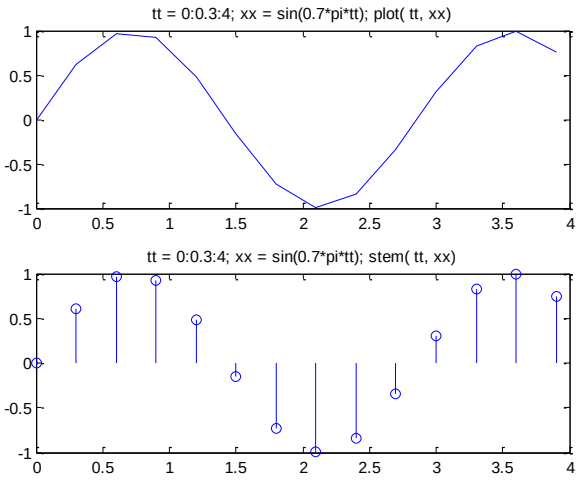
Operasi ini digunakan untuk menempelkan dua atau lebih array dengan syarat syarat tertetu sesuai dengan operasi concatenation yangdiinginkan. Dalam MATLAB terdapat dua buah fungsi yang dapat digunakan untuk melakukan proses concatenation (penempelan) arrays. Fungsi tersebut adalah vertcat dan horzcat. Penjelasan lanjut dapat dilihat pada help MATLAB untuk fungsi-fungsi tersebut.

2.3. PLOT DAN GRAFIK

MATLAB dapat menghasilkan plot dua dimensi x-y dan plot tiga dimensi, menayangkan citra, dan bahkan membuat dan memutar video. Dua fungsi yang yang sering digunakan pada praktikum ini adalah plot dan stem. Untuk memanggil fungsi ini, umumnya kita membutuhkan dua vektor (satu vektor juga bisa, namun untuk definisi yang berbeda, gunakan perintah help untuk melihat informasi yang lebih lengkap), untuk sumbu x dan sumbu y. Pemanggilan fungsi plot(x,y) akan menghasilkan suatu plot yang terkoneksi dengan garis lurus untuk setiap dua titik

{ (x(1),y(1), (x(2),y(2), (x(3),y(3), ……., (x(N),y(N) } seperti yang ditunjukkan pada gambar PA.1.

Pemanggilan fungsi stem(x,y) akan menghasilkan presentasi seperti yang ditunjukkan pada Gambar 1. 1 PA.1



Gambar 1. 1 PA.1

MATLAB memiliki banyak opsi *plotting* yang dapat dipelajari dengan help plotxy, help plotxyz, dan help graphics (versi 4) atau help graph2d, help graph3d, dan help specgraph (versi 5).

2.3.1. Figure Windows

Ketika MATLAB membuat sebuah plot, MATLAB menulis grafik tersebut ke *figure windows*. Anda bisa membuka beberapa *figure windows* namun setiap saat hanya satu *window* yang aktif. Setiap perintah plot pada *command window* akan mengalihkan keluarannya ke *window* yang aktif. Perintah figure(n) akan menampilkan sebuah *figure window* yang baru yang ditandai dengan bilangan n, atau membuatnya aktif kembali bila telah ada sebelumnya. Pengendalian terhadap berbagai atribut *window* (ukuran, lokasi, warna) juga mungkin dilakukan dengan perintah figure, yang melakukan inisialisasi terhadap *window plot*.

2.3.2. Mem-Plot beberapa grafik

Anda juga dapat membuat beberapa grafik/plot pada satu *window* dengan menggunakan fungsi subplot. Fungsi ini tidak melakukan proses *plotting*, namun hanya membagi window menjadi beberapa segmen. Sebagai contoh, perintah subplot(3,2,3) akan membagi *figure window* menjadi tiga baris dan dua kolom (jadi terdapat enam segmen) dan mengarahkan plot berikutnya ke segmen kiri baris kedua. Grafik pada PA.1 diperoleh dengan perintah subplot(2,1,1) dan subplot(2,1,2).

2.4. KONSTRUK PEMROGRAMAN

MATLAB mendukung paradigma pemrograman fungsional, di mana Anda dapat menyusun fungsi-fungsi secara *nested*. Perhatikan persamaan di bawah

$$\sum_{n=1}^{L} \log\left(\left|x_n\right|\right)$$

yang dapat diimplementaskan dengan hanya menggunakan satu baris kode MATLAB, yakni

```
sum( log( abs(x) ) )
```

di mana x adalah sebuah vektor yang berisi elemen-elemen x_n. Contoh ini mengilustrasikan MATLAB dalam bentuk yang paling efisien, di mana fungsi-fungsi individu dikombinasikan untuk menghasilkan keluaran. Penulisan kode-kode MATLAB yang efisien memerlukan gaya pemrograman yang menghasilkan fungsi-fungsi kecil yang divektorisasi. *Loop-loop* harus dihindari. Cara utama untuk menghindari *loop* adalah memanggil fungsi-fungsi *toolbox* sebanyak/sesering mungkin.

2.4.1. Fungsi-fungsi built-in MATLAB

Banyak fungsi-fungsi MATLAB yang dapat beroperasi pada skalar sama mudahnya dengan operasi pada *array*. Sebagai contoh, bila x adalah sebuah *array*, maka cos(x) mengembalikan sebuah *array* dengan ukuran yang sama seandainya x berisi kosinus dari setiap elemen x.

$$\cos(x) = \begin{bmatrix} \cos(x_{1,1}) & \cos(x_{1,2}) & \cdots & \cos(x_{1,n}) \\ \cos(x_{2,1}) & \cos(x_{2,2}) & \cdots & \cos(x_{2,n}) \\ \vdots & \vdots & \vdots & \vdots \\ \cos(x_{m,1}) & \cos(x_{m,2}) & \cdots & \cos(x_{m,n}) \end{bmatrix}$$

Perhatikan bahwa tidak ada *loop* yang diperlukan, meskipun cos(x) melakukan operasi kosinus pada setiap elemen *array*. Kebanyakan fungsi *transcendental* mengikuti aturan *pointwise* ini. Pada beberapa kasus khusus, adalah sangat penting untuk membedakan eksponensial matriks (expm) dengan eksponensial *pointwise*(exp):

$$\exp(A) = \begin{bmatrix} \exp(a_{1,1}) & \exp(a_{1,2}) & \cdots & \exp(a_{1,n}) \\ \exp(a_{2,1}) & \exp(a_{2,2}) & \cdots & \exp(a_{2,n}) \\ \vdots & \vdots & \vdots & \vdots \\ \exp(a_{m,1}) & \exp(a_{m,2}) & \cdots & \exp(a_{m,n}) \end{bmatrix}$$

2.4.2. Aliran Program (*Program Flow*)

Aliran program dapat dikendalikan pada MATLAB menggunakan pernyataan if, *loop*while, dan *loop*for. Pada MATLAB versi 5, terdapat juga pernyataan switch. Hal ini mirip dengan bahasa-bahasa tingkat tinggi seperti C++ atau PASCAL. Deskripsi dan contoh dari setiap konstruk program tersebut dapat dilihat dengan menggunakan perintah help.

2.5. MATLAB SCRIPTS

Setiap perintah/pernyataan yang dapat dimasukkan pada *window prompt* dapat disimpan pada sebuah *file* teks dan dieksekusi sebagai *script*. *File* teks tersebut dapat dibuat dengan menggunakan sembarang *editor* ASCII seperti program Notepad atau pada editor teks MATLAB. Ekstensi *file* harus berupa .m dan *script* tersebut dieksekusi pada MATLAB dengan hanya mengetikkan nama *file* (dengan atau tanpa ekstensi). Program-program tersebut umumnya dikenal dengan istilah *m-file*.Berikut merupakan contoh sebuah *m-file*:

```
tt = 0:0.3:4;
 xx = sin(0.7*pi*tt);
 subplot(2,1,1)
 plot( tt, xx)
 title('tt = 0:0.3:4; xx = sin(0.7*pi*tt); plot( tt, xx)')
 subplot(2,1,2)
 stem( tt, xx)
 title(''tt = 0:0.3:4; xx = sin(0.7*pi*tt); plot( tt, xx)')
```

Bila perintah-perintah ini disimpan dengan *file* bernama plotstem.m maka pengetikan plotstem pada *command prompt* akan menjalankan *file* tersebut, dan kedelapan baris perintah akan dieksekusi sama halnya bila mereka diketikkan baris per baris pada *command prompt*. Hasilnya adalah dua buah plot seperti yang tampak pada gambar A.1.

Setiap perintah/pernyataan yang dapat dimasukkan pada *window prompt* dapat disimpan pada sebuah *file* teks dan dieksekusi sebagai *script*. *File* teks tersebut dapat dibuat dengan menggunakan sembarang *editor* ASCII seperti program Notepad atau pada editor teks MATLAB. Ekstensi *file* harus berupa .m dan *script* tersebut dieksekusi pada MATLAB dengan hanya mengetikkan nama *file* (dengan atau tanpa ekstensi). Program-program tersebut umumnya dikenal dengan istilah *m-file*.Berikut merupakan contoh sebuah *m-file*:

```
tt = 0:0.3:4;
 xx = sin(0.7*pi*tt);
 subplot(2,1,1)
 plot( tt, xx)
 title('tt = 0:0.3:4; xx = sin(0.7*pi*tt); plot( tt, xx)')
 subplot(2,1,2)
 stem( tt, xx)
 title(''tt = 0:0.3:4; xx = sin(0.7*pi*tt); plot( tt, xx)')
```

Bila perintah-perintah ini disimpan dengan *file* bernama plotstem.m maka pengetikan plotstem pada *command prompt* akan menjalankan *file* tersebut, dan kedelapan baris perintah akan dieksekusi sama halnya bila mereka diketikkan baris per baris pada *command prompt*. Hasilnya adalah dua buah plot seperti yang tampak pada gambar A.1.

2.6. MENULIS FUNGSI MATLAB

Anda dapat menulis fungsi sendiri dan kemudian ditambahkan pada *environment* MATLAB. Fungsi-fungsi ini merupakan jenis lain dari *m-file*, dan dibuat sebagai sebuah *file* ASCII menggunakan *editor* teks. Kata pertama pada *m-file* tersebut haruslah *keyword*function untuk memberitahukan MATLAB bahwa *file* tersebut diperlakukan sebagai sebuah fungsi dengan argumen. Pada baris yang sama juga berisi *calling template* yang menyatakan argumen input dan output dari fungsi. Nama *file* untuk *m-file* tersebut haruslah berekstensi .m dan nama fungsi tersebut akan menjadi nama dari perintah baru pada MATLAB. Sebagai contoh, perhatikan *file* berikut, yang mengekstrak L buah elemen terakhir dari sebuah vektor

```
function y = foo( x, L )
 %FOO mengambil L buah titik terakhir dari x
 %   penggunaan:
 %                  y = foo( x, L )
 %   di mana:
 %                  x = vektor input
 %                  L = jumlah titik yang ingin diambil
 %                  y = vektor output
 N = length(x);
 If (L > N)
 error('vektor input terlalu pendek')
 end
 y = x(( N-L+1):N );
```

Bila *file* ini disimpan dengan nama foo.m, operasi ini dapat dipanggil dari MATLAB *command line* dengan cara mengetikkan

```
aa = foo( (1:2:37), 7 );
```

Outputnya akan berupa tujuh elemen terakhir dari vektor (1:2:37), yakni

```
aa = [  25   27   29   31   33   35   37  ]
```

2.6.1. Membuat sebuah fungsi *clip*

Kebanyakan fungsi dapat ditulis menurut format standar. Perhatikan sebuah fumgsi m-*file*clip yang mengambil dua buah argumen (sebuah vektor sinyal dan nilai skalar pembatas/*threshold*) dan mengembalikan sebuah vektor sinyal keluaran. Anda dapat menggunakan sebuah *editor* untuk menghasilkan *file* ASCII dari clip.m yang berisi pernyataan-pernyataan berikut.

Komentar-komentar ini akan ditampilkan bila Anda mengetikkan help clip

Langkah pertama adalah mencari dimensi matriks x

Input dapat berupa vektor kolom atau baris

Karena x adalah variabel lokal, kita dapat mengubahnya tanpa mempengaruhi workspace

Menghasilkan vektor output

```
function y = clip( x, Limit )
%CLIP mensaturasikan magnitud of x[n] pada Limit
%   ketika | x[n] | > Limit, buat | x[n] | = Limit
%
%   penggunaan: y = clip( x, Limit )
%
%   x           – vektor sinyal input
%   Limit       – nilai pembatas
%   y           – vektor output setelah clipping
[nrows ncols] = size(x);
if(ncols ~= 1 & nrows ~=1 )  %--bukan keduanya
error( 'CLIP: input bukan sebuah vektor')
end
Lx = max( [nrows ncols] );   %--Panjang
for n = 1: Lx     %--Loop seluruh vektor
if( abs( x( n ) ) > Limit )
x( n ) = sign( x ( n ) )*Limit;  %--saturasi
end     %--mempertahankan tanda x(n)
end
y = x;        %-- kopi ke vektor output
```

Kita dapat memecah *m-file*clip.m menjadi empat bagian:

1. Definisi input-output: Setiap *m-file* fungsi harus diawali kata function. Informasi yang mengikuti kata function pada baris yang sama merupakan deklarasi bagaimana fungsi

tersebut dipanggil dan argumen apa yang dibutuhkan. Nama **fungsi harus sama dengan nama *m-file***; bila terjadi konflik maka nama *m-file*-lah yang dikenali pada MATLAB *command environment*.

Argumen-argumen input ditulis di dalam tanda kurung setelah nama fungsi. Setiap input adalah sebuah matriks. Argumen output (yang juga berupa matriks) berada di sebelah kiri tanda =. Argumen output ganda juga dimungkinkan dengan menggunakan tanda kurung siku […], misalnya perintah size(x) mengembalikan jumlah baris dan kolom kedua variabel yang berbeda. Akhirnya, perhatikan bahwa tidak ada *command* eksplisit untuk mengembalikan sebuah output. MATLAB mengembalikan semua nilai yang terkandung pada matriks output ketika perhitungannya selesai dilakukan. Baris terakhir pada clip.m mengembalikan nilai y yang ter-*clip* ke y, jadi vektor yang ter-*clip* dikembalikan. MATLAB tidak memiliki perintah return, tapi ia langsung mengakhiri fungsi tersebut tanpa memerlukan argumen.

Perbedaan esensial antara *m-file* fungsi dan *m-filescripts* adalah *dummy variables* vs *permanent variables*. MATLAB menggunakan metode "*call by value*" sehingga fungsi tersebut membuat kopian lokal dari argumennya. Variabel-variabel lokal tersebut akan segera hilang saat fungsi tersebut selesai. Sebagai contoh, pernyataan berikut menghasilkan sebuah vektor wclipped dari vektor input ww.

```
>> wwclipped = clip(ww, 0.9999);
```

*Array*ww dan wwclipped adalah *permanent variables* pada MATLAB *workspace*. *Array* sementara yang dihasilkan di dalam clip.m (yaitu nrows, ncols, Lx dan n) hanya ada saat fungsi tersebut dijalankan. Mereka akan segera hilang ketika fungsi selesai dijalankan. Lebih lanjut, nama-nama variabel tersebut bersifat lokal (hanya bagi clip.m) sehingga nama varibel x juga dapat digunakan pada *workspace* sebagai *permanent variables*. Tentunya, ide-ide ini bukanlah sesuatu yang asing bagi Anda yang pernah mengenal bahasa seperti C, FORTRAN, atau PASCAL.

2. Dokumentasi. Setiap baris yang diawali dengan tanda % adalah baris komentar. Jadi, Anda dapat mengetikkan help clip dan komentar-komentar dari *m-file* Anda akan ditampilakn pada layar Anda. Format tersebut mengikuti "aturan" : nama fungsi, prosedur/ urutan pemanggilan, penjelasan singkat, dan definisi argumen input dan output.

3. Pengujian ukuran (*size*) dan kesalahan (*error*): Fungsi tersebut harus menguji dimensi dari matriks atau vektor yang akan dioperasikan. Informasi tersebut tidak perlu disampaikan sebagai argumen input yang terpisah, namun dapat diekstraks dengan fungsi size. Pada kasus fungsi clip.m, kita kita membatasi bahwa operasi hanya dapat dilakukan pada vektor, baik itu vektor baris 1 x L atau vektor kolom L x 1. Artinya, salah satu variabel tersebut haruslah sama dengan 1; kita mengakhiri fungsi tersebut dengan fungsi error, yang akan mencetak kalimat pada *command line* tersebut dan mengakhiri fungsi.

4. Operasi fungsi yang sebenarnya: Pada kasus fungsi clip.m, proses *clipping* dilakukan oleh loiop for, yang menguji besarnya nilai setiap elemen pada vektor x terhadap nilai

pembatas Limit. Pada kasus bilangan negatif, nilai yang di-*clip* akan diset ke –Limit, melalui perkalian dengan sign(x(n)). Ini mengasumsikan bahwa Limit dilewatkan sebagai sebuah bilangan positif, suatu fakta yang dapat diuji juga pada fase pengujian kesalahan.

Implementasi fungsi clip.m ini sangat tidak efisien karena adanya *loop*for. Pada sesi A.7.1 kita akan menunjukkan bagaimana men-vektorisasi program untuk meningkatkan efisiennya.

2.6.2. *Debugging* MATLAB m-*file*

Karena MATLAB adalah sebuah *environment* yang interaktif, *debugging* dapat dilakukan dengan cara menguji variabel-variabel pada *workspace*. MATLAB versi 4 dan 5 menyediakan *debugger* simbolik yang mendukung *breakpoints*. Karena fungsi yang berbeda dapat menggunakan nama variabel yang sama, adalah sangat penting untuk melacak konteks lokal ketika menguji variabel. Beberapa perintah *debugging* yang berguna didaftarkan di sini, dan yang lainnya dapat Anda temukan di help debug.

dbstop digunakan untuk mengeset sebuah *breakpoint* pada sebuah *m-file*. Ia juga dapat digunakan untuk memberikan sebuah peringatan ketika sebuah kesalahan terjadi dengan mengetikkan dbstop if error sebelum mengeksekusi *m-file* tersebut. Hal ini memungkinkan Anda menguji variabel di dalam fungsi dan *workspace* (dengan mengetikkan dbup)

dbstep akan mengembalikan sebuah peringatan (*prompt*) ketika setiap baris perintah dieksekusi.

dbcont menyebabkan sebuah eksekusi program yang normal berhenti, atau bila ada kesalahan, mengembalikan status Anda ke MATLAB *command prompt*.

dbquit menyebabkan Anda keluar dari *modedebug* dan kembali ke MATLAB *command prompt*.

keyboard dapat disisipkan ke *m-file* untuk menghentikan sementara eksekusi program, yang memberikan sebuah MATLAB *prompt* dalam bentuk K> untuk mengindikasikan bahwa itu bukan *command-line prompt*.

2.7. *TIPS* PEMROGRAMAN

Bagian ini akan memperkenalkan beberapa *tips* pemrograman yang akan meningkatkan kecepatan program MATLAB Anda. Untuk mengetahui lebih banyak tentang *tips* dan ide, perhatian gaya-gaya penulisan pada *m-file* (*built-in function*) yang tersedia pada *toolbox* MATLAB. Sebagai contoh, ketikkan perintah

```
type angle
type conv
type trapz
```

Mempelajari gaya pemrograman orang lain merupakan cara yang efisien untuk memahami suatu bahasa pemrograman komputer. Pada petunjuk-petunjuk berikut, kita akan membahas hal-hal yang terpenting dalam penulisan kode MATLAB yang baik. Petunjuk-petunjuk ini akan menjadi sangat berguna ketika Anda semakin mengenal MATLAB.

2.7.1. Menghindari *loop*

Karena MATLAB merupakan sebuah bahasa interpretasi (*interpreted language*), maka terdapat beberapa gaya pemrograman yang sangat tidak efisien. Salah satunya adalah penggunaan *loop*for untuk melakukan operasi sederhana terhadap sebuah matriks atau vektor. Bila dimungkinkan, Anda seharusnya mencoba mencari sebuah fungsi vektor (atau komposisi *nested* dari beberapa fungsi vektor) yang akan memberikan hasil yang sama, daripada menulis sebuah *loop*. Misalnya, bila operasi tersebut adalah penjumlahan semua elemen pada sebuah matriks, perbedaan antara memanggil fungsi sum dan menulis sebuah *loop* seperti pada bahasa FORTRAN dapat sangat mengejutkan; *loop* tersebut akan sangat lambat dalam menjalankan eksekusi akibat sifat *interpreted* dari MATLAB. Perhatikan tiga metode pada penjumlahan matriks berikut

Loop *ganda diperlukan untuk mengindeks semua elemen matriks*

```
[Nrows, Ncols] = size(x);
xsum = 0.0;
for m = 1: Nrows
for n = 1: Ncols
xsum = xsum + x(m,n);
```

sum menjumlahkan semua elemen setiap kolom

```
Xsum = sum( sum(x) );
```

x(:) adalah sebuah vektor yang berisi semua elemen matriks

```
Xsum = sum( x(:) );
```

Metode pertama adalah ekivalen MATLAB untuk pemrograman konvensional. Dua metode terakhir mengandalkan fungsi *built-in*sum, yang memiliki perbedaan karakteristik, tergantung dari apakah argumennya berupa sebuah matriks atau vektor (disebut operator *overloading*). Ketika beraksi pada sebuah matriks, sum mengembalikan sebuah vektor baris yang berisi penjumlahah kolom, dan ketika beraksi pada sebuah vektor baris (atau kolom), penjumlahan

tersebut adaah skalar. Pada metode ketiga, yang merupkan metode terefisien, matriks x dikonversikan ke vektor kolom dengan operator *colon* (:). Dan selanjutnya Anda hanya perlu memanggil fungsi sum sekali saja.

2.7.2. Pengulangan Baris atau Kolom

Sering kali kita perlu membentuk sebuah matriks dari sebuah vektor dengan cara mereplika vektor tersebut pada baris atau kolom matriks. Bila matriks tersebut memiliki nilai yang sama maka fungsi seperti ones(M,N) dan zeros(M,N) dapat digunakan. Tetapi untuk mereplika suatu vektot kolom x untuk menghasilkan sebuah matriks yang memiliki kolom-kolom yang identik, sebuah *loop* dapat dihindari dengan menggunakan operasi perkalian *outer-product matrix* yang telah dibahas pada sesi A2.2. Fragmen kode MATLAB berikut akan melakukan tugas tersebut untuk sebelas kolom

```
x = (12:-2:0)' ;          % tanda ' menunjukkan conjugatetranspose
X = x * ones(1,11)
```

Bila x adalah sebuah vektor kolom L, maka matrik X yang dihasilkan oleh *outer-product* tersebut adalah L x 11. Pada contoh ini, L = 7. Perhatikan bahwa penulisan pada MATLAB bersifat *case-sensitive*, jadi variabel x dan X adalah berbeda.

2.7.3. Vektorisasi operasi logika

Adalah mungkin untuk menvektorisasi program yang berisi pernyataan kondisional if, else. Fungsi clip pada A.2 merupakan contoh yang baik untuk mendemonstrasikan jenis vektorisasi ini. *Loop*for pada fungsi ini berisi sebuah pengujian logikal dan tidak tampak sebagai kandidat yang cocok untuk operasi vektor. Namun, operator logikal dan relasional pada MATLAB, seperti lebih besar dari, dapat diaplikasikan pada matriks. Sebagai contoh, pengujian "lebih besar dari" pada matriks 3 x 3 akan mengembalikan sebuah matriks 3 x 3 yang hanya berisi 1 dan 0.

```
>> x  = [ 1  2  –3;  3  –2  1; 4  0  –1]      % membuat matriks pengujian
    x  = [ 1   2   –3
           3   -2    1
           4    0    1 ]
```

```
>> mx  = x > 0                    % menguji kondisi lebih besar
   mx = [ 1   1  0
          1   0  1
          1   0  0]
```

```
>> y =  mx.* x                    % perkalian pointwise dengan matriks maksing mx
   y = [  1    2    0
          3    0    1
          4    0    0 ]
```

Elemen nol menunjukkan kondisi yang salah, dan elemen 1 menunjukkan kondisi yang benar. Jadi, ketika kita melakukan perkalian pointwise antara x dan matriks masking mx, kita memperoleh suatu matriks di mana semua elemen negatif diset ke nol. Perhatikan bahwa dua pernyataan terakhir memroses keseluruhan matriks tanpa menggunakan loop for. Karena saturasi yang dilakukan pada clip.m mensyaratkan kita mengubah nilai-nilai yang besar pada x, kita dapat mengimplementasikan keseluruhan loop for dengan tiga perkalian array. Hal ini mengantarkan kita ke operator saturasi tervektorisasi yang dapat bekerja pada matriks, maupun vektor sekaligus:

```
y = Limit*(x > Limit) – Limit*(x < -Limit) + x.*(abs(x) <= Limit);
```

Tiga matriks masking diperlukan untuk mewakili masing-masing kasus saturasi positif, saturasi negatif, dan tanpa aksi. Operasi penjumlahan berkoresponden dengan operator logika OR pada semua kasus ini. Jumlah operasi aritmetik yang harus dilakukan pada pernyataan ini adalah 3N perkalian dan 2N penjumlahan, di mana N adalah jumlah elemen pada x. Tampak proses ini memerlukan lebih banyak pekerjaan daripada loop pada clip.m bila kita hanya memperhitungkan operasi aritmetik. Namun, “harga” yang harus dibayar untuk interpretasi kode sangat mahal. Pernyataan tervektorisasi (vertorized) ini hanya diinterpretasikan sekali saja, sementara tiga pernyataan di dalam loop for harus direinterpretasikan sebanyak N kali. Bila Anda menggunakan etime untuk mencatat waktu kedua implementasi tersebut, maka versi vektorisasi akan jauh lebih cepat untuk vektor-vektor panjang.

2.7.4. Membuat sinyal *impulse*

Contoh sederhana lainnya diberikan oleh trik berikut untuk menghasilkan sebuah vektor sinyal *impulse*:

```
nn = [ -10 : 25];
impulse = (nn == 0);
```

Hasilnya dapat diplot dengan menggunakan stem(nn, impulse). Fragmen kode ini sangat baik untuk melukiskan esensi dari rumus matematika yang mendefinisikan bahwa sinyal *impulse* hanya ada untuk n = 0.

$$\delta[n] = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases}$$

2.7.5. Fungsi Find

Sebuah alternatif lain untuk *masking* adalah menggunakan fungsi find. Ini tidak berarti bahwa pendekatan ini lebih efisien, namun hanya berupa pendekatan lain. Fungsi find akan menentukan daftar indeks pada sebuah vektor di mana kondisinya adalah benar (persyaratan dipenuhi). Misalnya, find(x > Limit) akan mengembalikan daftar indeks di mana vektor tersebut lebih besar dari nilai Limit. Jadi, kita dapat melakukan saturasi sebagai berikut:

```
y = x
jkl = find(y > Limit)
y(jkl) = Limit*ones(size(jkl));
jkl = find(y < -Limit);
y(jkl) = -Limit*ones(size(jkl));
```

Fungsi ones diperlukan untuk menghasilkan sebuah vektor pada sisi kanan yang memiliki ukuran yang sama dengan jumlah elemen pada jkl. Pada versi 5, hal ini tidak diperlukan karena sebuah skalar yang di-*assign* ke vektor berarti di-*assign* pula untuk setiap elemen vektor.

2.7.6. Vektorisasi

Upaya "menghindari *loop*for" tidak selalu mudah untuk diikuti karena hal ini berarti algoritma harus disusun kembali dalam bentuk vektor. Bila notasi matriks-vektor dimasukkan dalam program MATLAB, maka kode tersebut akan berjalan jauh lebih cepat. Bahkan, *loop* untuk pengujian logika dapat divektorisasi bila *mask* diciptakan untuk semua kondisi. Jadi sebuah tujuan yang lebih masuk akal adalah

Eliminasi semua *loop*for

2.7.7. Gaya Pemrograman

Bila ada sebuah kalimat yang merangkum gaya pemrograman yang baik, maka bunyinya mungkin seperti berikut

Usahakan fungsi Anda pendek dan nama variabelnya panjang

Kalimat di atas sangat tepat untuk MATLAB. Setiap fungsi seharusnya memiliki satu tujuan khusus. Hal ini akan akan menghasilkan modul-modul yang singkat dan sederhana yang dapat di-*link* bersama dengan komposisi fungsional untuk menghasilkan operasi yang kompleks. Hindari kecenderungan untuk membuat fungsi super dengan banyak opsi dan keluaran.

MATLAB mendukung nama variabel hingga 32 variabel. Gunakan fitur ini untuk memberikan nama variabel yang deskriptif. Dengan cara ini, jumlah komentar-komentar yang "mengotori" kode Anda dapat dikurangi secara drastis. Komentar-komentar seharusnya hanya dibatasi untuk informasi help dan trik-trik yang digunakan pada kode tersebut.

3. PERSIAPAN PRAKTIKUM DAN TUGAS PENDAHULUAN

Pelajari penggunaan perintah `fir1`, `fir2`, `filter`, `freqz` dan perintah-perintah lain yang berhubungan dengan topik praktikum pada Matlab.

Tugas pendahuluan :

1. Jelaskan keuntungan penggunaan filter digital secara umum!
2. Apa yang dimaksud dengan filter FIR?
3. Jelaskan fungsi fir1 dan fir2 yang terdapat pada Matlab! Apa perbedaan antara keduanya?
4. Apa yang dimaksud dengan koefisien filter (yang diperoleh dari fungsi desain filter dari Matlab)?

4. PERCOBAAN

4.1. PERALATAN YANG DIGUNAKAN

1. 1 unit komputer
2. Software Matlab

4.2. PROSEDUR PRAKTIKUM

Sebelum praktikum dilaksanakan, lakukan beberapa hal berikut ini:

1. Pastikan komputer yang akan digunakan berfungsi dengan normal dan tidak ada masalah apapun.
2. Software Matlab sudah terinstal dalam komputer.

4.2.1. Percobaan membuat sinyal input filter berupa superposisi beberapa sinyal sinusoidal dengan frekuensi berbeda.

1. Pada Matlab, representasikan sinyal dalam vektor (matriks 1 x N, N panjang vektor). Kita akan merepresentasikan sumbu waktu dimana untuk $0<t<2\pi$, kita beri panjang vektor 100 (100 sampel) dengan perintah `>>i=1:100;`
2. Buat 3 sinyal sinusoidal pada frekuensi pencuplikan fs=16000 Hz untuk masing-masing frekuensi sinyal f1=200 Hz, f2=1000 Hz, f3=5000 Hz. Ketikan : >>sin1=sin(2*pi*i*f1/fs);sin2=sin(2*pi*i*f2/fs);sin3=sin(2*pi*i*f3/fs);
3. Jumlahkan ketiga sinyal tersebut menjadi satu sinyal sinusoidal rusak dengan perintah `>>sintot=(sin1+sin2+sin3)/3;`
4. Coba plot gambarnya dengan perintah `>>plot(sintot);`
5. Lihat juga respon frekuensinya dengan perintah `freqz`.
6. Kini Anda telah memiliki sinyal input untuk *filter* yang akan kita rancang

4.2.2. Percobaan desain dan simulasi filter FIR 1

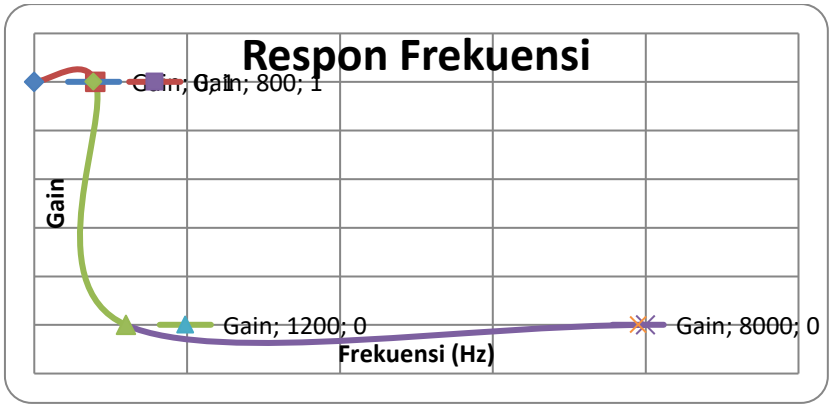
Kita akan coba beberapa *filter* FIR dengan spesifikasi berikut (frekuensi boleh diubah-ubah)

- Filter FIR *low-pass* orde 32 dengan frekuensi *cut-off* 800 Hz
- Filter FIR *band-pass* orde 32 dengan frekuensi *pass* 1000 – 3000 Hz
- Filter FIR *high-pass* orde 32 dengan frekuensi *cut-off* 6000Hz

1. Rancang ketiga *filter* di atas, cari koefisien *filter*-nya dengan perintah yang sesuai (`fir1`). Catat masing-masing koefisien *filter*.
2. Lihat frekuensi respon masing-masing *filter* dengan perintah `freqz`. Gambarkan hasilnya. Analisa pada frekuensi cut off.

4.2.3. Percobaan desain dan simulasi filter FIR 2

Kita akan coba mendesain *filter* FIR dengan metoda frekuensi sampling dengan respon frekuensi seperti pada Gambar 1. 2 Respon Frekuensi Filter



Gambar 1. 2 Respon Frekuensi Filter

1. Rancang *filter* di atas, cari koefisien *filter*-nya dengan perintah yang sesuai (`fir2`). Catat koefisien *filter*. Gunakan orde 16.

2. Lihat frekuensi respon *filter* dengan perintah `freqz`. Gambarkan hasilnya. Lakukan untuk orde lebih besar misalnya 128. Bandingkan dengan hasil sebelumnya.

4.2.4. Percobaan membuat m-file untuk melakukan pem-filter-an FIR

Pada bagian ini anda diminta untuk membuat m-file untuk melakukan pemfilteran FIR saja, untuk m-file anda tidak diperbolehkan memanggil fungsi internal MATLAB. Bandingkan hasilnya dengan percobaan dengan menggunakan perintah *filter* dari MATLAB. (Untuk kelancaran praktikum source code bisa dipersiapkan sebelum praktikum)

Catatan:

- Pada laporan m-file disertakan, dan berikan penjelasan algoritma yang anda gunakan untuk melakukan pem-filter-an FIR.

HINT: Koefisien filter FIR adalah merupakan respon impuls dari filter.

5. MENGAKHIRI PERCOBAAN

Sebelum keluar dari ruang praktikum, rapikan meja praktikumdan matikan computer dari jala-jala listrik.

Periksa lagi lembar penggunaan meja. Praktikan yang tidak menandatangani **lembar penggunaan meja** atau membereskan meja ketika praktikum berakhir akan mendapatkan **potongan nilai.**

Pastikan asisten telah menandatangani catatan percobaan kali ini pada Buku Catatan Laboratorium anda. Catatan percobaan yang tidak ditandatangani oleh asisten tidak akan dinilai.

CONTOH FORMAT LAPORAN

1.1. PERANCANGAN FILTER DENGAN MATLAB

Plot Sinyal Input

Sinyal Input	*Respon Frekuensi*

1.2. DESAIN FILTER FIR

Low-pass orde 31 f cut-off 800 Hz	*Koefisien Filter*
Respon Frekuensi Filter	
Band-pass orde 31 f cut-off 1000-3000 Hz	*Koefisien Filter*
Respon Frekuensi Filter	
High-pass orde 31 f cut-off 6000 Hz	*Koefisien Filter*
Respon Frekuensi Filter	

PERCOBAAN II

SIMULASI FILTER FIR REALTIME

Dev-C++ akan digunakan secara ekstensif pada praktikum ini. Modul ini memberikan tinjauan mengenai filter FIR realtime dapat diimplementasikan. Pada modul ini, praktikan akan membuat dan melakukan simulasi untuk implementasi filter realtime dengan menggunakan Dev-C++. Pada modul ini juga dikenalkan isu numerik yang dihadapi ketika menggunakan prosesor DSP fixed point.

1. TUJUAN

1. Mempelajari bagaimana mengimplementasikan dan melakukan simulasi filter FIR realtime menggunakan Dev-C++
2. Mengenal model bilangan fraksional untuk prosesor DSP fixed point.
3. Mengimplementasikan dan melakukan simulasi filter FIR realtime dengan bilangan fraksional menggunakan Dev-C++

2. DASAR TEORI

2.1. FILTER FIR REALTIME

Sistem realtime disebut juga dengan sistem waktu nyata. Sistem realtime harus dapat memberikan respon yang tepat dalam batasan waktu yang ditentukan. Realtime adalah metode realisasi, sehingga setiap tugas spesifik dapat dikerjakan pada waktu spesifik dengan siklus clock sistem.

Pada modul sebelumnya kita menggunakan filter FIR offline (non realtime). Terdapat pebedaaan di antara algoritma sistem realtime dan algoritma sistem offline. Ide dasar dari algoritma realtime adalah hanya input saat ini dan yang telah lewat yang tersedia. Kita tidak memiliki akses kepada input yang akan datang berikutnya. Pada modul sebelumnya fungsi filter yang digunakan memiliki akses kepada keseluruhan sampel pada sinyal input.

Algoritma filter realtime diperlukan untuk implementasi filter realtime dengan hardware DSP. Untuk keperluan ini maka diperlukan simulasi algoritma filter realtime-like dan dibandingkan hasilnya dengan algoritma filter offline.

Sistem DSP adalah sistem waktu diskrit yang mengambil sampel sinyal input dan menghasilkan sampel sinyal output dengan frekuensi sampling tertentu yang konstan. Artinya pada sistem DSP realtime, pada rentang waktu antar sampling, haruslah dapat dilakukan proses

perhitungan sampel output yang merupakan hasil pengolahan sinyal digital (bisa berupa filter). Artinya proses komputasi pengolahan sinyal digital untuk tiap sampel haruslah dapat diselesaikan sebelum sampel berikutnya diambil.

Karakteristik simulasi realtime-like adalah sebagai berikut:

1. Loop untuk kesesuaian dengan kelakuan clock
2. Akses hanya kepada input yang telah ada pada array input
3. Update output setiap siklus loop

2.2. ISU NUMERIK

Kebanyakan prosesor DSP merupakan prosesor fixed point. Prosesor floating point jauh lebih rumit dari prosesor fixed point sehingga membutuhkan harga yang lebih mahal. Untuk membuat sistem yang cost effective maka prosesor DSP direalisasikan dengan prosesor fixed point. Prosesor fixed point secara natural set intruksi dasar matematiknya hanya mendukung operasi bilangan bulat (integer)

Memahami isu numerik adalah sangat penting untuk memperoleh performansi terbaik dari sebuah prosesor fixed-point. Masalahnya adalah kebanyakan sinyal dan juga koefisien bernilai bilangan real. Sehingga operasi pengolahan sinyal merupakan operasi perkalian dan penjumlahan bilangan real. Namun DSP yang kita gunakan merupakan prosesor fixed point.

Dari percobaan sebelumnya Anda telah memperoleh koefisien *filter* untuk melakukan pem-*filter*-an. Kita akan menggunakannya sebagai koefisien *filter* yang akan diterapkan di BF561. Nantinya kita akan mencoba program *filter* dalam bahasa pemograman C ataupun Assembly, jadi kita dapat menuliskan koefisien tersebut di program kita, seperti berikut (contoh)

int koef={a1,a2,a3,a4};

Perhatikan bahwa koefisien harus ditulis dalam format Q32 dan jangan lupa bahwa prosesor yang kita gunakan adalah prosesor *fixed-point* 32 *bit* sehingga setiap bilangan direpresentasikan dalam 32 bit dengan kisaran -2^{-31} sampai $2^{31}-1$ atau –2147483648 – 2147483647 untuk –1 sampai 1.

Q31 adalah salah satu contoh format Q, yaitu penulisan bilangan untuk bilangan *fixed-point*. Pada BF561 yang merupakan prosesor *fixed-point* 32 bit, maka untuk N=32, digunakan format Q31. Format Q31 ini merepresentasikan fraksional 31 bit.

Pada format ini, MSB adalah *sign bit* yang diikuti oleh suatu titik imajiner, dan kemudian 31 bit atau mantisa yang dinormalisasi ke 1. Bilangan Q31 memiliki kisaran desimal antara –1 sampai 0.9999 (0x8000h sampai 0x7FFFh). Bilangan fraksional h dapat direpresentasikan sebagai $h = \sum_{n=0}^{N} b_n 2^{-n}$ atau $h= -b_0 2^0 + b_1 2^{-1} + b_2 2^{-2} + ... + b_N 2^{-N}$, dimana h adalah bilangan fraksional dan b adalah biner 1 atau 0.

Konversi bilangan real ke bilangan fraksional dilakukan dengan

```
koef=round(2^31*koef);
```

Untuk konversi bilangan fraksional ke bilangan real dilakukan dengan

```
koef=koef / 2^31;
```

3. PERSIAPAN PRAKTIKUM DAN TUGAS PENDAHULUAN

Pelajari proses pemfilteran dengan menggunakan buffering, circular buffer, dan operasi matematik dengan bilangan fraksional.

Tugas pendahuluan :

1. Bagaimana cara menggunakan koefisien filter agar dapat menghasilkan filter yang dapat memfilter sinyal masukan? (Jelaskan secara singkat dan melalui persamaan matematis)
2. Bagaimana cara mengimplementasikan hal tersebut pada program?
3. Buatlah flowchart program dari pertanyaan nomor 2 jika filter mempunyai 3 buah koefisien!
4. Apa yang dimaksud pemfilteran dengan buffer sirkular?
5. Siapkan *source code* percobaan 4.2.5 pada Arduino IDE dan MATLAB untuk kode pada Arduino IDE, kosongkan bagian *array* koef_filter[BUFFERLENGTH]!

4. PERCOBAAN

4.1. PERALATAN YANG DIGUNAKAN

1. 1 unit komputer
2. Software Matlab
3. Software Arduino IDE versi 1.8.19 ke atas dengan *library board* ESP32 versi 2.0.4
4. 1 unit ESP32 beserta kabel

4.2. PROSEDUR PRAKTIKUM

Sebelum praktikum dilaksanakan, lakukan beberapa hal berikut ini:

1. Pastikan komputer yang akan digunakan berfungsi dengan normal dan tidak ada masalah apapun.
2. Software Dev-C++ sudah terinstal dalam komputer.
3. Software Arduino IDE sudah terinstal dalam komputer.

Hal yang perlu diperhatikan ketika menjalankan simulasi adalah selalu buat file yang berbeda dengan file yang sedang dibuka oleh Microsoft Office Excel agar simulasi dapat menulis file. Karena file yang sedang dibuka tidak akan bisa ditulis oleh program simulasi.

4.2.1. Percobaan membuat m-file untuk simulasi pem-filter-an realtime FIR

Pada bagian ini anda diminta untuk membuat simulasi pem-filter-an realtime FIR dengan Dev-C++. Fungsi internal filter menggunakan metoda non realtime, karena menghitung sinyal output dengan sinyal input yang lengkap. Buat perhitungan filter realtime FIR dengan menggunakan Dev-C++.

Ketentuan:

- Gunakan buffer input sepanjang orde filter (Contoh: jika orde filter adalah 16 maka buffer input adalah 16)
- Gunakan metoda loop sejumlah sampel sinyal input untuk simulasi clock
- Pada setiap loop (clock) mengindikasikan adanya sampel input yang baru. Ambil input baru dan tempatkan pada buffer input
- Pada setiap loop (clock) hitunglah sampel sinyal output yang dihasilkan pada loop tersebut, lalu tempatkan pada array output

Jalankan software Dev-C++. Pilih File > New Project. Kemudian pilih Jenis Console Application. Ganti nama project sesuai keinginan anda misalkan FIR. Pilih OK. Kemudian keluar jendela baru dan pilih folder untuk menempatkan project yang akan dibuat. Akan otomatis dibuat file source code main.cpp. Edit file ini dengan source code berikut:

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <math.h>

using namespace std;

// panjang tap filter
#define BUFFERLENGTH 9
#define LENGTH 100

#define PI 3.14159265

float koef_filter[BUFFERLENGTH] = {0.020041, 0.064976, 0.167492,
0.250847, 0.250847, 0.167492, 0.064976, 0.020041};
float buffer[BUFFERLENGTH];
float x[LENGTH];
float y[LENGTH];

int i, j;

float filter(float input) {
      float hasil;
      //buffering
```

```
	for(j = BUFFERLENGTH - 1; j > 0; j--) {
		buffer[j] = buffer[j-1];
	}
	//input disimpan di buffer 0
	buffer[0] = input;

	// perhitungan filter
	hasil = 0;
	for(j = 0; j < BUFFERLENGTH; j++) {
		hasil = hasil + buffer[j] * koef_filter[j];
	}

	// kembalikan hasil pemfilteran
	return hasil;
}

int main(int argc, char *argv[])
{
    FILE *pFile;

    //siapkan input
    for(i=0; i < LENGTH; i++) {
            x[i] =
(sin(2*PI*200/16000*i)+sin(2*PI*1000/16000*i)+sin(2*PI*5000/16000*i)
) / 3;
    }

    //inisialisasi buffer dengan nol
    for(i=0; i < BUFFERLENGTH; i++) {
            buffer[i] = 0;
    }

    //loop mengambil input tiap sampel dan menaruhnya pada output
    for(i=0; i < LENGTH; i++) {
            y[i] = filter(x[i]);
    }

    //tulis output ke file
    pFile = fopen("output.txt", "w");
    for(i=0; i < LENGTH; i++) {
            fprintf(pFile,"%4.4f\n", y[i]);
    }
    fclose(pFile);

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Perhatikan ada bagian untuk menyiapkan input, bagian pemfilteran dan hasil output ditulis ke file dengan format text. Edit koefisien filter sesuai dengan filter yang didesain dengan MATLAB. Jalankan simulasi maka akan dihasilkan file text. Buka file text ini dengan Microsoft Office Excel. Lalu buat grafiknya.

Bandingkan hasil simulasi ini dengan fungsi internal filter.

4.2.2. Percobaan membuat m-file untuk simulasi pem-filter-an realtime FIR dengan buffer sirkular

Pada bagian ini anda diminta untuk memodifikasi hasil dari percobaan sebelumnya dengan menggunakan metode buffering sirkular. Dengan metoda ini maka setiap kali didapatkan sampel input, maka sampel-sampel terdahulu pada buffer tidak perlu digeser. Metoda buffering sirkular ini sangat sering digunakan karena operasi write pada memori biasanya membutuhkan waktu eksekusi yang cukup lama dibandingan dengan operasi matematik biasa.

Edit source code dari bagian sebelumnya dengan mengubah bagian buffering dan perhitungan filter. Lakukan simulasi dan buatlah grafiknya.

Bandingkan hasil simulasi ini dengan hasil sebelumnya.

4.2.3. Percobaan membuat m-file untuk simulasi pem-filter-an realtime FIR dengan bilangan fraksional

Pada bagian ini anda diminta untuk memodifikasi hasil dari percobaan sebelumnya dengan menggunakan bilangan fraksional 32 bit. Semua koefisien filter direpresentasikan dengan bilangan fraksional. Buffer diisi dengan sampel input baru yang diisi dengan sampel input yang telah dikonversikan dari bilangan real menjadi bilangan fraksional. Modifikasi perkalian bilangan real menjadi perkalian bilangan fraksional. Simpan sampel sinyal output yang dihasilkan dengan terlebih dahulu mengkonversikan bilangan fraksional menjadi bilangan real.

Edit source code dari bagian sebelumnya dengan mengubah fungsi filter, juga koefisien dan buffer menjadi tipe integer. Tambahkan bagian konversi output menjadi float sebelum ditulis ke file. (Gunakan casting).

Bandingkan hasil simulasi ini dengan hasil sebelumnya.

4.2.4. Simulasi *Hardware in The Loop* pem-filter-an realtime FIR pada ESP32

Pada bagian ini, anda diminta untuk membuat simulasi *hardware in the loop* implementasi filter FIR *realtime* pada ESP32. Digunakan Software Arduino IDE untuk *flashing* kode filter FIR ke ESP32. Dalam menjalankan simulasi, software MATLAB digunakan untuk mengirim data sinyal masukan ke ESP32 dan menerima sinyal keluaran hasil filter dari ESP32 melalui komunikasi serial.

1. Buka sofware Arduino IDE, buat sketch baru dengan klik File > New.
2. Salin kode dibawah dan simpan dengan nama FIR.ino, lakukan modifikasi pada bagian panjang *buffer* sesuai dengan panjang orde filter yang dirancang. Kemudian isi koefisien filter FIR low-pass orde 32 dengan frekuensi cut-off 800 Hz pada *array* koef_filter yang didapat dari modul sebelumnya.

```
#define BUFFERLENGTH 33

// koefisien filter (dari MATLAB)
const float koef_filter[BUFFERLENGTH]= { ...
};

float buffer[BUFFERLENGTH];
int i,j = 0;
float input, output;

float filter(float input) {
    float hasil;
    //buffering
    for(j = BUFFERLENGTH - 1; j > 0; j--) {
        buffer[j] = buffer[j-1];
    }
    //input disimpan di buffer 0
    buffer[0] = input;
    // perhitungan filter
    hasil = 0;
    for(j = 0; j < BUFFERLENGTH; j++) {
    hasil = hasil + buffer[j] * koef_filter[j];
    }
    // kembalikan hasil pemfilteran
    return hasil;
    }

void setup() {
  //Inisialisasi serial
  Serial.begin(230400);
  //Inisialisasi buffer
  for(i=0; i < BUFFERLENGTH; i++) {
    buffer[i] = 0;
    }
}
```

```
void loop(){
        if (Serial.available()>0){
           input = Serial.parseFloat();
           output = filter(input);
           Serial.println(output,4);
           }
}
```

3. Sambungkan ESP32 ke komputer. Pastikan ESP32 terdeteksi oleh komputer, pilih board dengan membuka Tools>Board>ESP32 Arduino>ESP Dev Module. Serta pilih port serial dengan membuka Tools>Port>pilih port ESP32 (COM ...).

4. *Compile* dan *Upload* kode FIR.ino yang telah dibuat dengan klik Upload pada Arduino IDE. Selagi mengupload, tekan tombol Boot pada ESP32 agar komputer dapat mendeteksi ESP32.

5. Buka software MATLAB, buat script baru dengan nama HILS.m. Salin kode berikut.

```
close all; clear all;
%Buka Serial Port
ESP32_PORT = serialport("COM3", 230400);

%Buat Sinyal Sinusoid
i=0:99;
%Isi fungsi sin
sin1= ...;
sin2= ...;
sin3= ... ;
sintot = ... ;

%Bulatkan ke 4 desimal
sintot_rounded= round(sintot*10000)/10000;

%deklarasi array penyimpan data serial
global filt_sig;

enable = 1;
j = 1;
```

```
figure;

%looping
while(1)
    if (j <= 100)
        %Penulisan nilai sinyal berikutnya ke ESP32 melalui serial
port
        if (enable == 1)
            writeline(ESP32_PORT, num2str(sintot_rounded(j),4));
            enable = 0;
        end

        %Pembacaan hasil filter sinyal dari serial port
        if (ESP32_PORT.NumBytesAvailable > 0)
            filt_sig(j) = str2double(readline(ESP32_PORT));
            %Lakukan plotting data yang didapat
            plot(filt_sig);
            j = j + 1;
            enable = 1;
        end
        pause(0.0001);
    else
            break;
      end

end
```

6. Modifikasi kode HILS.m pada bagian pembuatan sinyal sinusoid dengan 3 sinyal sinusoidal pada frekuensi pencuplikan fs=16000 Hz untuk masing-masing frekuensi sinyal f1=200 Hz, f2=1000 Hz, f3=5000 Hz. Bila diperlukan modifikasi fungsi serialport() jika port ESP32 yang terdeteksi di komputer tidak sesuai.

7. Jalankan kode MATLAB dengan klik Run, akan muncul sinyal keluaran hasil filter dari ESP32. Gambar sinyal yang didapatkan pada BCL.

8. Ulangi langkah percobaan untuk beberapa variasi filter.

    - Filter FIR band-pass orde 32 dengan frekuensi pass 1000 – 3000 Hz
    - Filter FIR high-pass orde 32 dengan frekuensi cut-off 6000Hz
    - Filter FIR low-pass orde 16 dengan frekuensi cut-off 800 Hz

9. Lakukan analisis hasil yang dibuat di laporan.

4.2.5. Simulasi *Hardware in The Loop* pem-filter-an realtime FIR pada ESP32 dengan Buffer Sirkular

Pada bagian ini anda diminta untuk memodifikasi hasil dari percobaan sebelumnya dengan menggunakan metode buffering sirkular. Edit kode **FIR.ino** dari bagian sebelumnya dengan mengubah bagian buffering dan perhitungan filter.

Lakukan simulasi Hardware in The Loop dengan ESP32 untuk filter FIR low-pass orde 32 dengan frekuensi cut-off 800 Hz.

4.2.6. Simulasi *Hardware in The Loop* pem-filter-an realtime FIR pada ESP32 dengan Bilangan Fraksional

Pada bagian ini anda diminta untuk memodifikasi hasil dari percobaan sebelumnya dengan menggunakan bilangan fraksional 32 bit. Edit **FIR.ino** dari bagian sebelumnya dengan mengubah fungsi filter, juga koefisien dan buffer menjadi tipe integer. Tambahkan bagian konversi output menjadi float sebelum dikirim melalui Serial. (Gunakan casting).

Lakukan simulasi Hardware in The Loop dengan ESP32 untuk filter FIR low-pass orde 32 dengan frekuensi cut-off 800 Hz.

5. MENGAKHIRI PERCOBAAN

Sebelum keluar dari ruang praktikum, rapikan meja praktikumdan matikan computer dari jala-jala listrik.

Periksa lagi lembar penggunaan meja. Praktikan yang tidak menandatangani **lembar penggunaan meja** atau membereskan meja ketika praktikum berakhir akan mendapatkan **potongan nilai.**

Pastikan asisten telah menandatangani catatan percobaan kali ini pada Buku Catatan Laboratorium anda. Catatan percobaan yang tidak ditandatangani oleh asisten tidak akan dinilai.