

Introducción a *Octave*

Facundo Gutiérrez

Elementos de Cálculo Numérico

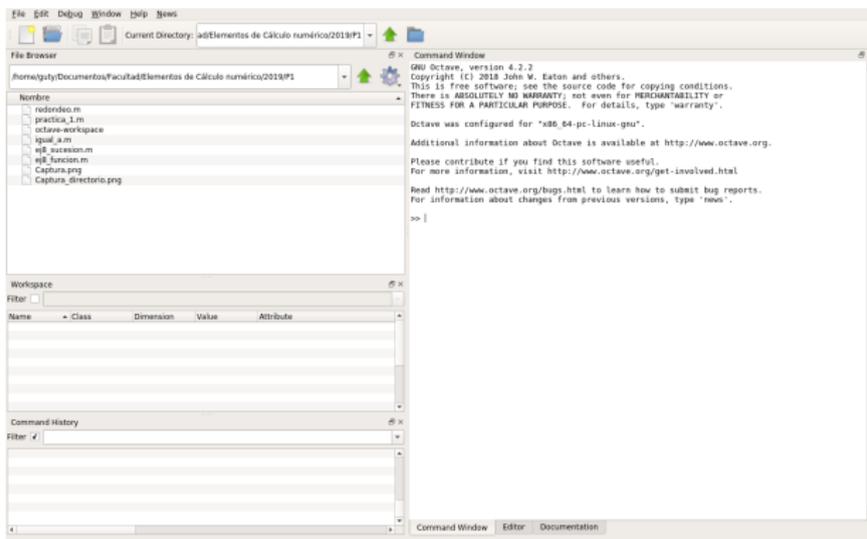
- 1 Primeros pasos
 - Instalación
 - Comprendiendo la interfaz gráfica
- 2 Comandos básicos
 - Operaciones básicas
 - Variables
 - Vectores
- 3 Generar gráficos
- 4 Sentencias **for**, **if** y **while**
- 5 Trabajando con funciones
 - Uso de `function_handle`
 - Funciones creadas por el usuario
- 6 Algunos tips

Instalando Octave

- En Windows: <https://www.gnu.org/software/octave>
- En Linux: En la terminal:
sudo apt-add-repository ppa:octave/stable
sudo apt-get update
sudo apt-get install octave

Instalando Octave

- En Windows: <https://www.gnu.org/software/octave>
- En Linux: En la terminal:
sudo apt-add-repository ppa:octave/stable
sudo apt-get update
sudo apt-get install octave



Interfaz gráfica

The screenshot displays the Octave graphical user interface. The top menu bar includes File, Edit, Debug, Window, and Help. The File Browser on the left shows the current directory as 'ad/Elementos de Cálculo numérico/2019/P1'. The Workspace panel contains a table with the following data:

Name	Class	Dimension	Value	Attribute
y	double	1x100	[-1.6097, -2.1...	
x	double	1x100	[-5, -2.9394, ...	
v	double	1x10	[1, 2, 3, 4, 5, ...	
ans	double	1x1	12	
a	double	1x1	6	

The Command History panel shows the following commands and their outputs:

```
ans
# Octave 4.0.0, Tue Mar 19 17:05:34 2019 -03
print(2)
for i = 1:10
  ans
  for i = 1:10
    2^i
    ans = 2
    ans = 4
    ans = 6
    ans = 8
    ans = 10
    ans = 2
    ans = 6
    ans = 8
    ans = 10
    ans = 12
  end
end
ans = 2
ans = 4
ans = 6
ans = 8
ans = 10
ans = 2
ans = 6
ans = 8
ans = 10
ans = 12
end
>> |
```

The Editor window shows a script named 'practica_1.m' with the following code:

```
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

The Command Window shows the following commands and their outputs:

```
line: 158 col: 4 encoding: UTF-8 eol: LF
Command Window
>> y = 2*sin(4*x);
>> plot(x,y);
>> y = x*sin(4*x);
error: operator *: nonconformant arguments (op1 is 1x100, op2 is 1x100)
>> y = x.*sin(4*x);
>> plot(x,y);
>> x = linspace(-3,3,100);
>> y = x.*sin(4*x);
>> plot(x,y);
>> v = [1 2 3 4 5 1 3 4 5 6];
>> for a = v
    2^a
end
ans = 2
ans = 4
ans = 6
ans = 8
ans = 10
ans = 2
ans = 6
ans = 8
ans = 10
ans = 12
end
>> |
```

Interfaz gráfica

The screenshot displays the Octave graphical user interface (GUI) with several key components:

- File Browser:** Located at the top left, it shows the current directory as `adElementos de Cálculo numérico\2019\P1`. The file list includes `ej1_funcion.m`, `ej2_funcion.m`, `ej3_1.m`, `practica_1.m`, and `redondeo.m`.
- Workspace:** A table below the file browser showing variables in the workspace:

Name	Class	Dimension	Value	Attribute
y	double	1x100	[-1.6097, -2.1...	
x	double	1x100	[-5, -2.9394, ...	
v	double	1x10	[1, 2, 3, 4, 5, ...	
ans	double	1x1	12	
a	double	1x1	6	
- Command History:** Shows the execution of a script named `practica_1.m`, including commands like `ans = 2`, `ans = 4`, `ans = 6`, `ans = 8`, `ans = 10`, `ans = 2`, `ans = 6`, `ans = 8`, `ans = 10`, `ans = 12`, and `ans = 1`.
- Command Window:** Displays the current command being executed:

```
>> y = 2*sin(4*x);
>> plot(x,y);
>> y = x*sin(4*x);
error: operator *: nonconformant arguments (op1 is 1x100, op2 is 1x100)
>> y = x.*sin(4*x);
>> plot(x,y);
>> x = linspace(-3,3,100);
>> [xy] = x.*sin(4*x);
>> plot(x,y);
>> v = [1 2 3 4 5 1 3 4 5 6];
>> for a = v
    2*a
end
```

Directorio actual: Indica en qué carpeta estamos trabajando actualmente (importa al trabajar con funciones).

Interfaz gráfica

The screenshot displays the Octave graphical user interface. The top menu bar includes File, Edit, Debug, Window, and Help. The current directory is set to 'ad/Elementos de Cálculo numérico/2019/P1'. The File browser on the left shows a tree view of files, with 'practica_3.m' selected. The Workspace window shows a table of variables:

Name	Class	Dimension	Value	Attribute
y	double	1x100	[-1.6097, -2.1...	
v	double	1x100	[-5, -2.9394, ...	
ans	double	1x1	12	
a	double	1x1	6	

The Command History window shows the following commands:

```
ans
# Octave 4.0.0, Tue Mar 19 17:05:34 2019 -03
print(2)
for i = 1:10
  ans
endfor
2^i
ans = 4
5^4
ans = 2
A = 5;
B = 9;
ans
A*B
ans = 12
f = @(x,y) sqrt(1+x^2+y^2);
```

The Command Window shows the execution of the script 'practica_3.m':

```
line: 158 col: 4 encoding: UTF-8 eol: LF
Command Window
>> y = 2*i*sin(4*x);
>> plot(x,y)
>> y = x*i*sin(4*x);
error: operator *: nonconformant arguments (op1 is 1x100, op2 is 1x100)
>> y = x.*sin(4*x);
>> plot(x,y)
>> x = linspace(-3,3,100);
>> y = x.*sin(4*x);
>> plot(x,y)
>> v = [1 2 3 4 5 1 3 4 5 6];
>> for a = v
  2*a
end
ans = 2
ans = 4
ans = 6
ans = 8
ans = 10
ans = 2
ans = 6
ans = 8
ans = 10
ans = 12
>> |
```

Explorador de archivos: Nos permite ubicarnos en el directorio que deseamos.

Interfaz gráfica

The screenshot shows the Octave GUI interface. The main editor window displays a script named 'practica_1.m' with the following code:

```
148
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

The Command Window shows the following output:

```
ans = 2
ans = 4
ans = 6
ans = 8
ans = 10
ans = 2
ans = 6
ans = 8
ans = 8
ans = 10
ans = 12
>> |
```

The Workspace window is highlighted with a red box and contains the following table:

Name	Class	Dimension	Value	Attribute
y	double	1x100	[-1.6097, -2.1...	
x	double	1x100	[-5, -2.9394, ...	
v	double	1x10	[1, 2, 3, 4, 5, ...	
ans	double	1x1	12	
a	double	1x1	6	

The Command History window shows the following commands:

```
ans
# Octave 4.0.0, Tue Mar 19 17:05:34 2019 -03
print(2)
for i = 1:10
endfor
ans = 2
ans = 4
ans = 6
ans = 8
ans = 10
ans = 2
ans = 6
ans = 8
ans = 8
ans = 10
ans = 12
| = @(x,y) sqrt(1+x^2+y^2)
|
```

Espacio de trabajo: Nos indica qué variables ya tienen un valor asignado (y su valor).

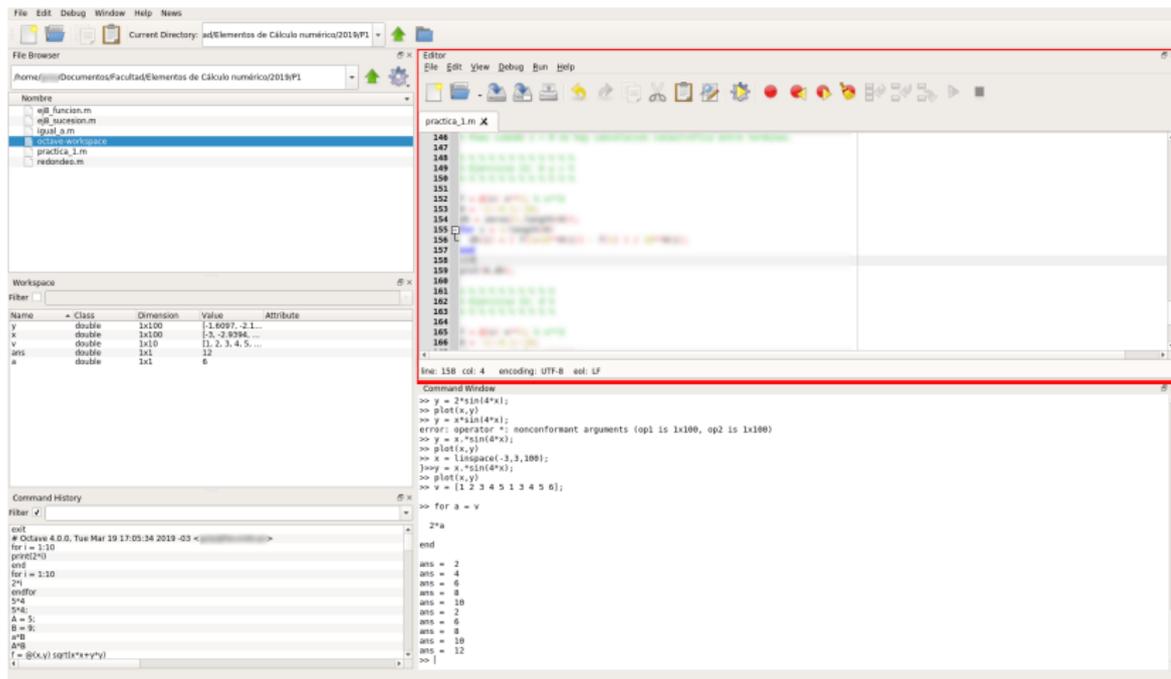
Interfaz gráfica

The screenshot displays the Octave GUI interface. The Command History window is highlighted with a red border and contains the following text:

```
Command History
Fiber v
ans
# Octave 4.0.0, Tue Mar 19 17:05:34 2019 -03
print(2)
for i = 1:10
  v(i)
endfor
2%
ans = 1
ans = 2
ans = 3
ans = 4
ans = 5
ans = 6
ans = 7
ans = 8
ans = 9
ans = 10
ans = 11
ans = 12
ans = 13
ans = 14
ans = 15
ans = 16
ans = 17
ans = 18
ans = 19
ans = 20
ans = 21
ans = 22
ans = 23
ans = 24
ans = 25
ans = 26
ans = 27
ans = 28
ans = 29
ans = 30
ans = 31
ans = 32
ans = 33
ans = 34
ans = 35
ans = 36
ans = 37
ans = 38
ans = 39
ans = 40
ans = 41
ans = 42
ans = 43
ans = 44
ans = 45
ans = 46
ans = 47
ans = 48
ans = 49
ans = 50
ans = 51
ans = 52
ans = 53
ans = 54
ans = 55
ans = 56
ans = 57
ans = 58
ans = 59
ans = 60
ans = 61
ans = 62
ans = 63
ans = 64
ans = 65
ans = 66
ans = 67
ans = 68
ans = 69
ans = 70
ans = 71
ans = 72
ans = 73
ans = 74
ans = 75
ans = 76
ans = 77
ans = 78
ans = 79
ans = 80
ans = 81
ans = 82
ans = 83
ans = 84
ans = 85
ans = 86
ans = 87
ans = 88
ans = 89
ans = 90
ans = 91
ans = 92
ans = 93
ans = 94
ans = 95
ans = 96
ans = 97
ans = 98
ans = 99
ans = 100
```

Historial de comandos: Almacena los comandos que fueron ingresados por el usuario.

Interfaz gráfica



Editor: Nos permite escribir listas de comandos que serán ejecutadas secuencialmente.

Interfaz gráfica

The screenshot displays the Octave graphical user interface. At the top, there is a menu bar with 'File', 'Edit', 'Debug', 'Window', and 'Help'. Below the menu bar, the 'File Browser' shows the current directory as 'ad/Elementos de Cálculo numérico/2019/P1'. The 'Workspace' panel on the left contains a table with the following data:

Name	Class	Dimension	Value	Attribute
y	double	1x100	[-1.6097, -2.1...	
x	double	1x100	[-5, -2.9394, ...	
ans	double	1x10	[1, 2, 3, 4, 5, ...	
a	double	1x1	12	

The 'Command History' panel shows the following commands:

```
ans  
# Octave 4.0.0, Tue Mar 19 17:05:34 2019 -03  
print(2)  
for i = 1:10  
end  
ans =  
2%  
endfor  
5%  
A = 5;  
B = 9;  
ans =  
4%  
A*B  
[ = @(x,y) sqrt(1+x*y);  
ans =
```

The 'Command Window' (highlighted with a red border) shows the following commands and their outputs:

```
Command Window  
>> y = 2*sin(4*x);  
>> plot(x,y)  
>> y = x*sin(4*x);  
error: operator *: nonconformant arguments (op1 is 1x100, op2 is 1x100)  
>> y = x.*sin(4*x);  
>> plot(x,y)  
>> x = linspace(-3,3,100);  
>> y = x.*sin(4*x);  
>> plot(x,y)  
>> v = [1 2 3 4 5 1 3 4 5 6];  
ans =  
1 2 3 4 5 1 3 4 5 6  
>> for a = v  
2*a  
end  
ans =  
2  
ans =  
4  
ans =  
6  
ans =  
8  
ans =  
10  
ans =  
2  
ans =  
6  
ans =  
8  
ans =  
10  
ans =  
12  
>> |
```

Ventana de comandos: Aquí se imprimen los resultados a los comandos que fueron ingresados.

Operadores básicos

- $+$ para la suma

Operadores básicos

- $+$ para la suma
- $-$ para la resta

Operadores básicos

- `+` para la suma
- `-` para la resta
- `*` para la multiplicación y `.*` para la multiplicación *elemento a elemento* (toma sentido al trabajar con vectores o matrices)

Operadores básicos

- `+` para la suma
- `-` para la resta
- `*` para la multiplicación y `.*` para la multiplicación *elemento a elemento* (toma sentido al trabajar con vectores o matrices)
- `^` o `**` para la potenciación y `.^` o `.**` para la potenciación *elemento a elemento* (ídem)

Operadores básicos

- `+` para la suma
- `-` para la resta
- `*` para la multiplicación y `.*` para la multiplicación *elemento a elemento* (toma sentido al trabajar con vectores o matrices)
- `^` o `**` para la potenciación y `.^` o `.**` para la potenciación *elemento a elemento* (ídem)
- `/` para la división y `./` para la división *elemento a elemento* (ídem)

Operadores básicos

- `+` para la suma
- `-` para la resta
- `*` para la multiplicación y `.*` para la multiplicación *elemento a elemento* (toma sentido al trabajar con vectores o matrices)
- `^` o `**` para la potenciación y `.^` o `.**` para la potenciación *elemento a elemento* (ídem)
- `/` para la división y `./` para la división *elemento a elemento* (ídem)

Ingresar en la *ventana de comandos*:

`(1.5+2)*0.5`

Operadores básicos

- `+` para la suma
- `-` para la resta
- `*` para la multiplicación y `.*` para la multiplicación *elemento a elemento* (toma sentido al trabajar con vectores o matrices)
- `^` o `**` para la potenciación y `.^` o `.**` para la potenciación *elemento a elemento* (ídem)
- `/` para la división y `./` para la división *elemento a elemento* (ídem)

Ingresar en la *ventana de comandos*:

```
(1.5+2)**0.5
```

Operadores básicos

- `+` para la suma
- `-` para la resta
- `*` para la multiplicación y `.*` para la multiplicación *elemento a elemento* (toma sentido al trabajar con vectores o matrices)
- `^` o `**` para la potenciación y `.^` o `**.` para la potenciación *elemento a elemento* (ídem)
- `/` para la división y `./` para la división *elemento a elemento* (ídem)

Ingresar en la *ventana de comandos*:

`(1.5-2)/0.5`

Variables

Podemos almacenar el resultado de una expresión en una *variable*.

No solo **nos evita tener que tipear una misma expresión** repetidas veces, también hace que lo que escribimos sea **más claro**.

Variables

Podemos almacenar el resultado de una expresión en una *variable*.

No solo **nos evita tener que tipear una misma expresión** repetidas veces, también hace que lo que escribimos sea **más claro**.

La sintaxis para *asignarle una expresión a una variable* funciona de la siguiente forma:

- `variable = expresion` `variable ← expresion`

Variables

Podemos almacenar el resultado de una expresión en una *variable*.

No solo **nos evita tener que tipear una misma expresión** repetidas veces, también hace que lo que escribimos sea **más claro**.

La sintaxis para *asignarle una expresión a una variable* funciona de la siguiente forma:

- `variable = expresion` `variable ← expresion`

Ingresar en la *ventana de comandos*:

```
var_1 = (1.5+2)*0.5
```

Variables

Podemos almacenar el resultado de una expresión en una *variable*.

No solo **nos evita tener que tipear una misma expresión** repetidas veces, también hace que lo que escribimos sea **más claro**.

La sintaxis para *asignarle una expresión a una variable* funciona de la siguiente forma:

- `variable = expresion` `variable ← expresion`

Ingresar en la *ventana de comandos*:

```
var_2 = (1.5+2)**0.5
```

Variables

Podemos almacenar el resultado de una expresión en una *variable*.

No solo **nos evita tener que tipear una misma expresión** repetidas veces, también hace que lo que escribimos sea **más claro**.

La sintaxis para *asignarle una expresión a una variable* funciona de la siguiente forma:

- `variable = expresion` `variable ← expresion`

Ingresar en la *ventana de comandos*:

```
resultado = var_1+var_2
```

Variables

Podemos almacenar el resultado de una expresión en una *variable*.

No solo **nos evita tener que tipear una misma expresión** repetidas veces, también hace que lo que escribimos sea **más claro**.

La sintaxis para *asignarle una expresión a una variable* funciona de la siguiente forma:

- `variable = expresion` `variable ← expresion`

Ingresar en la *ventana de comandos*:

```
format long
```

Variables

Podemos almacenar el resultado de una expresión en una *variable*.

No solo **nos evita tener que tipear una misma expresión** repetidas veces, también hace que lo que escribimos sea **más claro**.

La sintaxis para *asignarle una expresión a una variable* funciona de la siguiente forma:

- `variable = expresion` `variable ← expresion`

Ingresar en la *ventana de comandos*:

```
resultado
```

Variables

Podemos almacenar el resultado de una expresión en una *variable*.

No solo **nos evita tener que tipear una misma expresión** repetidas veces, también hace que lo que escribimos sea **más claro**.

La sintaxis para *asignarle una expresión a una variable* funciona de la siguiente forma:

- `variable = expresion` `variable ← expresion`

Ingresar en la *ventana de comandos*:

```
format short
```

Variables

Podemos almacenar el resultado de una expresión en una *variable*.

No solo **nos evita tener que tipear una misma expresión** repetidas veces, también hace que lo que escribimos sea **más claro**.

La sintaxis para *asignarle una expresión a una variable* funciona de la siguiente forma:

- `variable = expresion` `variable ← expresion`

Ingresar en la *ventana de comandos*:

```
resultado
```

Vectores

Hay distintas formas de crear un vector en *Octave*.

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:

$$v = [1.5, 3.1, -3.4, 4.8]$$
$$v \leftarrow (1.5, 3.1, -3.4, 4.8)$$

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:

```
v = [1.5, 3.1, -3.4, 4.8]
```

```
v ← (1.5, 3.1, -3.4, 4.8)
```

- Usando el operador *dos puntos* (:)

```
v = comienzo:paso:final
```

$v_{i+1} = \text{comienzo} + i \cdot \text{paso}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{\text{final} - \text{comienzo}}{\text{paso}} \rfloor$

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:

```
v = [1.5, 3.1, -3.4, 4.8]
```

```
v ← (1.5, 3.1, -3.4, 4.8)
```

- Usando el operador *dos puntos* (:)

```
v = comienzo:paso:final
```

$$v_{i+1} = \text{comienzo} + i \cdot \text{paso}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{\text{final} - \text{comienzo}}{\text{paso}} \rfloor$$

- Usando funciones de *Octave*:

```
v = linspace(comienzo, final, n)
```

$$v_{i+1} = \text{comienzo} + i \cdot \frac{\text{final} - \text{comienzo}}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$$

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:

```
v = [1.5, 3.1, -3.4, 4.8]
```

```
v ← (1.5, 3.1, -3.4, 4.8)
```

- Usando el operador *dos puntos* (:)

```
v = comienzo:paso:final
```

$$v_{i+1} = \text{comienzo} + i \cdot \text{paso}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{\text{final} - \text{comienzo}}{\text{paso}} \rfloor$$

- Usando funciones de *Octave*:

```
v = linspace(comienzo, final, n)
```

$$v_{i+1} = \text{comienzo} + i \cdot \frac{\text{final} - \text{comienzo}}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$$

Ingresar en la *ventana de comandos*:

```
v = [1.5, 2, -3.5, 5]
```

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:

```
v = [1.5, 3.1, -3.4, 4.8]
```

```
v ← (1.5, 3.1, -3.4, 4.8)
```

- Usando el operador *dos puntos* (:)

```
v = comienzo:paso:final
```

$$v_{i+1} = \text{comienzo} + i \cdot \text{paso}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{\text{final} - \text{comienzo}}{\text{paso}} \rfloor$$

- Usando funciones de *Octave*:

```
v = linspace(comienzo, final, n)
```

$$v_{i+1} = \text{comienzo} + i \cdot \frac{\text{final} - \text{comienzo}}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$$

Ingresa en la *ventana de comandos*:

```
v(3)
```

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:

```
v = [1.5, 3.1, -3.4, 4.8]
```

```
v ← (1.5, 3.1, -3.4, 4.8)
```

- Usando el operador *dos puntos* (:)

```
v = comienzo:paso:final
```

$$v_{i+1} = \text{comienzo} + i \cdot \text{paso}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{\text{final} - \text{comienzo}}{\text{paso}} \rfloor$$

- Usando funciones de *Octave*:

```
v = linspace(comienzo, final, n)
```

$$v_{i+1} = \text{comienzo} + i \cdot \frac{\text{final} - \text{comienzo}}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$$

Ingresar en la *ventana de comandos*:

```
v+2
```

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:

```
v = [1.5, 3.1, -3.4, 4.8]
```

```
v ← (1.5, 3.1, -3.4, 4.8)
```

- Usando el operador *dos puntos* (:)

```
v = comienzo:paso:final
```

$$v_{i+1} = \text{comienzo} + i \cdot \text{paso}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{\text{final} - \text{comienzo}}{\text{paso}} \rfloor$$

- Usando funciones de *Octave*:

```
v = linspace(comienzo, final, n)
```

$$v_{i+1} = \text{comienzo} + i \cdot \frac{\text{final} - \text{comienzo}}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$$

Ingresar en la *ventana de comandos*:

```
v*2
```

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:

```
v = [1.5, 3.1, -3.4, 4.8]
```

```
v ← (1.5, 3.1, -3.4, 4.8)
```

- Usando el operador *dos puntos* (:)

```
v = comienzo:paso:final
```

$$v_{i+1} = \text{comienzo} + i \cdot \text{paso}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{\text{final} - \text{comienzo}}{\text{paso}} \rfloor$$

- Usando funciones de *Octave*:

```
v = linspace(comienzo, final, n)
```

$$v_{i+1} = \text{comienzo} + i \cdot \frac{\text{final} - \text{comienzo}}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$$

Ingresar en la *ventana de comandos*:

```
v**2
```

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:

```
v = [1.5, 3.1, -3.4, 4.8]
```

```
v ← (1.5, 3.1, -3.4, 4.8)
```

- Usando el operador *dos puntos* (:)

```
v = comienzo:paso:final
```

$$v_{i+1} = \text{comienzo} + i \cdot \text{paso}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{\text{final} - \text{comienzo}}{\text{paso}} \rfloor$$

- Usando funciones de *Octave*:

```
v = linspace(comienzo, final, n)
```

$$v_{i+1} = \text{comienzo} + i \cdot \frac{\text{final} - \text{comienzo}}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$$

Ingresar en la *ventana de comandos*:

```
v.**2
```

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:

```
v = [1.5, 3.1, -3.4, 4.8]
```

```
v ← (1.5, 3.1, -3.4, 4.8)
```

- Usando el operador *dos puntos* (:)

```
v = comienzo:paso:final
```

$$v_{i+1} = \text{comienzo} + i \cdot \text{paso}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{\text{final} - \text{comienzo}}{\text{paso}} \rfloor$$

- Usando funciones de *Octave*:

```
v = linspace(comienzo, final, n)
```

$$v_{i+1} = \text{comienzo} + i \cdot \frac{\text{final} - \text{comienzo}}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$$

Ingresar en la *ventana de comandos*:

```
u = -1:0.75:1
```

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:

```
v = [1.5, 3.1, -3.4, 4.8]
```

```
v ← (1.5, 3.1, -3.4, 4.8)
```

- Usando el operador *dos puntos* (:)

```
v = comienzo:pasos:final
```

$$v_{i+1} = \text{comienzo} + i \cdot \text{pasos}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{\text{final} - \text{comienzo}}{\text{pasos}} \rfloor$$

- Usando funciones de *Octave*:

```
v = linspace(comienzo, final, n)
```

$$v_{i+1} = \text{comienzo} + i \cdot \frac{\text{final} - \text{comienzo}}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$$

Ingresa en la *ventana de comandos*:

```
v+u
```

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:

```
v = [1.5, 3.1, -3.4, 4.8]
```

```
v ← (1.5, 3.1, -3.4, 4.8)
```

- Usando el operador *dos puntos* (:)

```
v = comienzo:paso:final
```

$$v_{i+1} = \text{comienzo} + i \cdot \text{paso}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{\text{final} - \text{comienzo}}{\text{paso}} \rfloor$$

- Usando funciones de *Octave*:

```
v = linspace(comienzo, final, n)
```

$$v_{i+1} = \text{comienzo} + i \cdot \frac{\text{final} - \text{comienzo}}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$$

Ingresar en la *ventana de comandos*:

```
v*u
```

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:

```
v = [1.5, 3.1, -3.4, 4.8]
```

```
v ← (1.5, 3.1, -3.4, 4.8)
```

- Usando el operador *dos puntos* (:)

```
v = comienzo:paso:final
```

$$v_{i+1} = \text{comienzo} + i \cdot \text{paso}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{\text{final} - \text{comienzo}}{\text{paso}} \rfloor$$

- Usando funciones de *Octave*:

```
v = linspace(comienzo, final, n)
```

$$v_{i+1} = \text{comienzo} + i \cdot \frac{\text{final} - \text{comienzo}}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$$

Ingresar en la *ventana de comandos*:

```
v.*u
```

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:

```
v = [1.5, 3.1, -3.4, 4.8]
```

```
v ← (1.5, 3.1, -3.4, 4.8)
```

- Usando el operador *dos puntos* (:)

```
v = comienzo:paso:final
```

$$v_{i+1} = \text{comienzo} + i \cdot \text{paso}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{\text{final} - \text{comienzo}}{\text{paso}} \rfloor$$

- Usando funciones de *Octave*:

```
v = linspace(comienzo, final, n)
```

$$v_{i+1} = \text{comienzo} + i \cdot \frac{\text{final} - \text{comienzo}}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$$

Ingresa en la *ventana de comandos*:

```
linspace(-1, 2, 10)
```

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:

```
v = [1.5, 3.1, -3.4, 4.8]
```

```
v ← (1.5, 3.1, -3.4, 4.8)
```

- Usando el operador *dos puntos* (:)

```
v = comienzo:paso:final
```

$$v_{i+1} = \text{comienzo} + i \cdot \text{paso}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{\text{final} - \text{comienzo}}{\text{paso}} \rfloor$$

- Usando funciones de *Octave*:

```
v = linspace(comienzo, final, n)
```

$$v_{i+1} = \text{comienzo} + i \cdot \frac{\text{final} - \text{comienzo}}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$$

Ingresar en la *ventana de comandos*:

```
linspace(-1, 2, 10);
```

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:

```
v = [1.5, 3.1, -3.4, 4.8]
```

```
v ← (1.5, 3.1, -3.4, 4.8)
```

- Usando el operador *dos puntos* (:)

```
v = comienzo:paso:final
```

$$v_{i+1} = \text{comienzo} + i \cdot \text{paso}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{\text{final} - \text{comienzo}}{\text{paso}} \rfloor$$

- Usando funciones de *Octave*:

```
v = linspace(comienzo, final, n)
```

$$v_{i+1} = \text{comienzo} + i \cdot \frac{\text{final} - \text{comienzo}}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$$

Ingresar en la *ventana de comandos*:

```
1:10
```

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:

```
v = [1.5, 3.1, -3.4, 4.8]
```

```
v ← (1.5, 3.1, -3.4, 4.8)
```

- Usando el operador *dos puntos* (:)

```
v = comienzo:paso:final
```

$$v_{i+1} = \text{comienzo} + i \cdot \text{paso}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{\text{final} - \text{comienzo}}{\text{paso}} \rfloor$$

- Usando funciones de *Octave*:

```
v = linspace(comienzo, final, n)
```

$$v_{i+1} = \text{comienzo} + i \cdot \frac{\text{final} - \text{comienzo}}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$$

Ingresar en la *ventana de comandos*:

```
v(2:3)
```

Hay distintas formas de crear un vector en *Octave*.

- Listar sus elementos:

```
v = [1.5, 3.1, -3.4, 4.8]
```

```
v ← (1.5, 3.1, -3.4, 4.8)
```

- Usando el operador *dos puntos* (:)

```
v = comienzo:paso:final
```

$$v_{i+1} = \text{comienzo} + i \cdot \text{paso}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i \leq \lfloor \frac{\text{final} - \text{comienzo}}{\text{paso}} \rfloor$$

- Usando funciones de *Octave*:

```
v = linspace(comienzo, final, n)
```

$$v_{i+1} = \text{comienzo} + i \cdot \frac{\text{final} - \text{comienzo}}{n-1}, \quad \forall i \in \mathbb{N}_0 \text{ tal que } 0 \leq i < n$$

Ingresar en la *ventana de comandos*:

```
[v, u]
```

Para graficar una función utilizamos el comando `plot`. En principio toma *dos vectores como argumentos*.

Grafica los *pares ordenados* formados por cada coordenada del primer vector y su correspondiente del segundo vector (deben ingresarse vectores con el mismo tamaño). Sin ninguna opción extra, se interpola linealmente entre pares consecutivos.

Para graficar una función utilizamos el comando `plot`. En principio toma *dos vectores como argumentos*.

Grafica los *pares ordenados* formados por cada coordenada del primer vector y su correspondiente del segundo vector (deben ingresarse vectores con el mismo tamaño). Sin ninguna opción extra, se interpola linealmente entre pares consecutivos.

```
x = linspace(-3,3,100)
```

```
y = x.*sin(4.*x)     $y_i = x_i \cdot \text{sen}(4x_i)$ 
```

```
plot(x,y)    Grafica los puntos  $(x_i, y_i)$ 
```

Sentencia for

Podemos utilizar este comando para recorrer un vector y ejecutar una misma serie de comandos a medida que se recorre el vector (usualmente se utiliza un vector de índices).

Sentencia for

Podemos utilizar este comando para recorrer un vector y ejecutar una misma serie de comandos a medida que se recorre el vector (usualmente se utiliza un vector de índices).

```
for elemento = vector elemento ∈ vector
```

```
    serie de comandos
```

```
end
```

Sentencia for

Podemos utilizar este comando para recorrer un vector y ejecutar una misma serie de comandos a medida que se recorre el vector (usualmente se utiliza un vector de índices).

```
for elemento = vector elemento ∈ vector
```

```
    serie de comandos
```

```
end
```

Si tenemos varios vectores de largo n , un patrón común es:

```
for i = 1:n  $i \in (1, \dots, n)$ 
```

```
    serie de comandos
```

```
end
```

Expresiones lógicas

- `<` menor que $2 < 3 \rightarrow 1$ (*True*)

Expresiones lógicas

- \lt menor que $2 < 3 \rightarrow 1$ (*True*)
- \gt mayor que $2 > 3 \rightarrow 0$ (*False*)

Expresiones lógicas

- \lt menor que $2 \lt 3 \rightarrow 1$ (*True*)
- \gt mayor que $2 \gt 3 \rightarrow 0$ (*False*)
- \leq menor o igual $2 \leq 3 \rightarrow 1$ (*True*)

Expresiones lógicas

- $<$ menor que $2 < 3 \rightarrow 1$ (*True*)
- $>$ mayor que $2 > 3 \rightarrow 0$ (*False*)
- $<=$ menor o igual $2 <= 3 \rightarrow 1$ (*True*)
- $>=$ mayor o igual $2 >= 3 \rightarrow 0$ (*False*)

Expresiones lógicas

- $<$ menor que $2 < 3 \rightarrow 1$ (*True*)
- $>$ mayor que $2 > 3 \rightarrow 0$ (*False*)
- $<=$ menor o igual $2 <= 3 \rightarrow 1$ (*True*)
- $>=$ mayor o igual $2 >= 3 \rightarrow 0$ (*False*)
- $==$ igual que $2 == 3 \rightarrow 0$ (*False*)

Expresiones lógicas

- `<` menor que $2 < 3 \rightarrow 1$ (*True*)
- `>` mayor que $2 > 3 \rightarrow 0$ (*False*)
- `<=` menor o igual $2 <= 3 \rightarrow 1$ (*True*)
- `>=` mayor o igual $2 >= 3 \rightarrow 0$ (*False*)
- `==` igual que $2 == 3 \rightarrow 0$ (*False*)
- `~=` distinto que $2 ~= 3 \rightarrow 1$ (*True*)

Expresiones lógicas

- $<$ menor que $2 < 3 \rightarrow 1(True)$
- $>$ mayor que $2 > 3 \rightarrow 0(False)$
- $<=$ menor o igual $2 <= 3 \rightarrow 1(True)$
- $>=$ mayor o igual $2 >= 3 \rightarrow 0(False)$
- $==$ igual que $2 == 3 \rightarrow 0(False)$
- $\sim=$ distinto que $2 \sim= 3 \rightarrow 1(True)$

Si queremos expresar que varias condiciones ocurran simultáneamente (conocido como el operador and), utilizamos $\&$ entre las condiciones.

Si queremos expresar que solo alguna de varias condiciones valga (conocido como el operador or), utilizamos $|$ (*barra vertical*) entre las condiciones.

Si queremos negar una condición, podemos utilizar \sim

Sentencia `if`

Nos sirve para ejecutar una serie de comandos específica en el caso de que ocurra la condición que queremos.

Sentencia if

Nos sirve para ejecutar una serie de comandos específica en el caso de que ocurra la condición que queremos.

```
if condicion_1
```

```
    serie de comandos 1
```

```
elseif condicion_2    opcional
```

```
    serie de comandos 2
```

```
else    opcional
```

```
    serie de comandos 3
```

```
end
```

Sentencia `while`

Mientras que se cumpla una cierta condición se ejecutará una serie de comandos específica.

Sentencia `while`

Mientras que se cumpla una cierta condición se ejecutará una serie de comandos específica.

```
while condicion  
    serie de comandos  
end
```

Sentencia `while`

Mientras que se cumpla una cierta condición se ejecutará una serie de comandos específica.

```
while condicion  
  
    serie de comandos  
  
end
```

Observación: Al salir del `while` necesariamente deja de valer `condicion`

Funciones de una línea

Para no repetir una misma expresión cambiando solo unos parámetros dentro de ella podemos utilizar funciones.

Funciones de una línea

Para no repetir una misma expresión cambiando solo unos parámetros dentro de ella podemos utilizar funciones.

`f = @(x) x.*sin(4.*x)` $f(x) = x \cdot \text{sen}(4x)$

Funciones de una línea

Para no repetir una misma expresión cambiando solo unos parámetros dentro de ella podemos utilizar funciones.

$$f = @(x) x.*\sin(4.*x) \quad f(x) = x \cdot \text{sen}(4x)$$

También funciona si tenemos más de un parámetro.

Funciones de una línea

Para no repetir una misma expresión cambiando solo unos parámetros dentro de ella podemos utilizar funciones.

```
f = @(x) x.*sin(4.*x)    f(x) = x · sen(4x)
```

También funciona si tenemos más de un parámetro.

```
f = @(x,y,z) sqrt(x.**2 + y.**2 + z.**2)
```

$$f(x, y, z) = \sqrt{x^2 + y^2 + z^2}$$

Funciones creadas por el usuario

Si tenemos que crear una función más sofisticada, que realiza varias operaciones con los parámetros para llegar al resultado, es conveniente crear una función en un archivo, por ejemplo `nombre.m`.

Funciones creadas por el usuario

Si tenemos que crear una función más sofisticada, que realiza varias operaciones con los parámetros para llegar al resultado, es conveniente crear una función en un archivo, por ejemplo `nombre.m`.

El archivo `nombre.m` deberá tener la siguiente forma:

```
function [res1,res2,...,resk] = nombre(arg1,...,argm)
```

serie de comandos que puede usar arg_1, \dots, arg_m

```
end
```

Funciones creadas por el usuario

Si tenemos que crear una función más sofisticada, que realiza varias operaciones con los parámetros para llegar al resultado, es conveniente crear una función en un archivo, por ejemplo `nombre.m`.

El archivo `nombre.m` deberá tener la siguiente forma:

```
function [res1,res2,...,resk] = nombre(arg1,...,argm)
```

serie de comandos que puede usar arg_1, \dots, arg_m

```
end
```

Al finalizar de correr la serie de comandos secuencialmente, retornará los valores que estén guardados en las variables res_1, \dots, res_k .

Para utilizar una función, debemos **estar situados en la misma carpeta donde está guardada** (ya sea desde otro archivo en el *editor* o en la *ventana de comandos*), y llamarla de la siguiente forma:

```
[var1,var2,...,vark] = nombre(arg1,...,argm)
```



Funciones creadas por el usuario

Si tenemos que crear una función más sofisticada, que realiza varias operaciones con los parámetros para llegar al resultado, es conveniente crear una función en un archivo, por ejemplo `nombre.m`.

El archivo `nombre.m` deberá tener la siguiente forma:

```
function [res1,res2,...,resk] = nombre(arg1,...,argm)
```

serie de comandos que puede usar arg_1, \dots, arg_m

```
end
```

Al finalizar de correr la serie de comandos secuencialmente, retornará los valores que estén guardados en las variables res_1, \dots, res_k .

Para utilizar una función, debemos **estar situados en la misma carpeta donde está guardada** (ya sea desde otro archivo en el *editor* o en la *ventana de comandos*), y llamarla de la siguiente forma:

```
[var1,var2,...,vark] = nombre(arg1,...,argm) Importante    
```

Algunos tips

- Trabajar en el editor.
- Podemos agregar comentarios con el símbolo `%` o el símbolo `#`.
- Guardar el trabajo continuamente (`CTRL + S` guarda directamente).
- Si tipeamos `help nombre_funcion` Octave nos detalla cómo utilizar la función en cuestión.
- Podemos ejecutar un conjunto de líneas seleccionadas apretando `F9`.
- Podemos guardar y ejecutar un script del editor con la tecla `F5`.
- Cuando algo que escribimos no funciona Octave nos detalla por qué (en la *ventana de comandos*).
- En la ventana de comandos, usando la flecha `↑` podemos visitar los últimos comandos ingresados.
- Para no escribir $*10^{38}$, podemos escribir `e38` (38 es un ejemplo).
- Con `CTRL + R` podemos comentar la línea actual, y descomentarla con `CTRL + SHIFT + R`.
- Indentar correctamente (alinear correctamente dónde comienza y termina cada sentencia) es muy importante.



John W. Eaton, David Bateman, Soren Hauberg, Rik Wehbring (2019)

Documentación oficial

<https://octave.org/doc/octave-5.1.0.pdf>



José María Valiente Cifuentes (2006)

Manual de iniciación a GNU Octave

http://softlibre.unizar.es/manuales/aplicaciones/octave/manual_octave.pdf