
| PART 1 |

Affichage de différentes valeurs avec disp :

1.

0.3

1.00D+100

2.061D-09

1.4142136

Affichage de e = sqrt(2) avec mprintf :
e = 1.4142136
e = 1.414213562373
e = 1.41421356237309515

Affichage de b = 0.3 avec mprintf :
b = 0.3000000
b = 0.300000000000
b = 0.29999999999999999
b = 0.29999999999999998889776975374843459576368331909179687500 0

Affichage de b et c avec la fonction 'ecrire' disponible dans nombres.sce

b =
2.99999999999999988897769753748434595763683319091796875e-01

c =
1.00000000000000021016978033233282513878227153874641880321881666098873
60023982790799717755191065313 28e+100

| PART 2 |

Codage de réels sur 64 bits :

Codage de -101 :
Codage 64 chiffres binaires :
signe s :
codage = 1
valeur = 1
exposant e :
codage = 10000000101
valeur = 1029
mantisse m :
codage = 1001010000000000000000000000000000000000000000000000
valeur = 2603643534573568

Codage de 0.3 :
Codage 64 chiffres binaires :
signe s :
codage = 0
valeur = 0
exposant e :
codage = 01111111101
valeur = 1021
mantisse m :
codage = 0011001100110011001100110011001100110011001100110011
valeur = 900719925474099

Codage de 0 :
Codage 64 chiffres binaires :
signe s :
codage = 0
valeur = 0
exposant e :
codage = 00000000000
valeur = 0
mantisse m :
codage = 0000000000000000000000000000000000000000000000000000
valeur = 0

Codage de 1 :
Codage 64 chiffres binaires :
signe s :
codage = 0
valeur = 0
exposant e :
codage = 01111111111
valeur = 1023
mantisse m :

codage = 0000000000000000000000000000000000000000000000000000000000000000
valeur = 0

Codage de 2 :
Codage 64 chiffres binaires :
signe s :
codage = 0
valeur = 0
exposant e :
codage = 10000000000
valeur = 1024
mantisse m :
codage = 0000000000000000000000000000000000000000000000000000000000000000
valeur = 0

Codage de 0.5 :
Codage 64 chiffres binaires :
signe s :
codage = 0
valeur = 0
exposant e :
codage = 01111111110
valeur = 1022
mantisse m :
codage = 0000000000000000000000000000000000000000000000000000000000000000
valeur = 0

Codage de 0.3 :
Codage 64 chiffres binaires :
signe s :
codage = 0
valeur = 0
exposant e :
codage = 01111111101
valeur = 1021
mantisse m :
codage = 0011001100110011001100110011001100110011001100110011001100110011
valeur = 900719925474099

Codage de 0.6 :
Codage 64 chiffres binaires :
signe s :
codage = 0
valeur = 0
exposant e :

codage = 01111111110
valeur = 1022
mantisse m :
codage = 0011001100110011001100110011001100110011001100110011001100110011
valeur = 900719925474099

Codage de 0.15 :
Codage 64 chiffres binaires :
signe s :
codage = 0
valeur = 0
exposant e :
codage = 01111111100
valeur = 1020
mantisse m :
codage = 0011001100110011001100110011001100110011001100110011001100110011
valeur = 900719925474099

.....

Decodage de plusieurs nombres binaires avec 'valeur_reel64' disponible dans nombres.sce

ex 1 : valeur attendue v = 1
1.e+00

ex 2 : valeur attendue v ~ 5.10^(-324)
4.9406564584124654417656879286822137236505980261432476442558568
2500675
5072702087518652998363616359923797965646954457177309266567103559
9397963
9877479601078187812630071319031140452784581716784898210368871863
3605699
8730723050006387409153564984387312473397273169615140031715385398
8074126
2385655911710266585566867681870395603106249319452715914924553293
3054565
4440112748012970999954193198940908041656332452475714786901472678
8015935
5238611550134803526493472019379026810710749170333222684475333572
2083243
1936092382893458368060106011506169809753078342277318329247904982
2524730
7763759272478746560847782037344696995336470179726777175851256605
5511991
3150489110145103786273816725095583738973359899366480994116420578

0263709
027924276754456522908753868250641971826553344726562 5e-324

ex 3 : valeur attendue v ~ 2,22.10^(-308)
2.225073858507201383090232717332404064219215980462331830553327 4
1688720
443481391819585428315901251102056406733973103581100515243416155
3460108
856012385377718821130777993532002330479610147442583636071921565
0469425
037342083752508066506166581589487204911799685916396485006359087
7011830
487479978088775374994945158045160505091539985658247081864511353
7935804
992115981085766051992433352114352390148795699609591288891602992
6415110
634663133936634775865130293717620473256317814856643508721228286
3764204
484681140761391147706280168985324411002416144742161856716615054
0154285
084716752901903161322778896729707373123334086988983175067838846
9260927
739779728586596549410913690954061364675687023986783152906809846
1721092
4625396728515625e-308

ex 4 : valeur attendue v ~ 1,79.10^308
1.797693134862315708145274237317043567980705675258449965989174 7
6803157
260780028538760589558632766878171540458953514382464234321326889
4641827
684675467035375169860499105765512820762454900903893289440758685
0845513
394230458323690322294816580855933212334827479782620414472316873
8177180
919299881250404026184124858368e+308

ex 5 : valeur attendue v = 1
1.e+00

ex 6 : valeur attendue v ~ 1
1.000000000000000222044604925031308084726333618164062 5e+00

.....

Mise en evidence de la precision machine :

Codage de 4.00000000000 :
Codage 64 chiffres binaires :
signe s :
codage = 0
valeur = 0
exposant e :
codage = 10000000001
valeur = 1025
mantisse m :
codage = 0000000000000000000000000000000000000000000000000000
valeur = 0

Codage de 4.1 = 4.0 + 10^(- 1) :
Codage 64 chiffres binaires :
signe s :
codage = 0
valeur = 0
exposant e :
codage = 10000000001
valeur = 1025
mantisse m :
codage = 0000011001100110011001100110011001100110011001100110
valeur = 112589990684262

Codage de 4.01 = 4.0 + 10^(- 2) :
Codage 64 chiffres binaires :
signe s :
codage = 0
valeur = 0
exposant e :
codage = 10000000001
valeur = 1025
mantisse m :
codage = 0000000101000111101011100001010001111010111000010
valeur = 11258999068426

Codage de 4.001 = 4.0 + 10^(- 3) :
Codage 64 chiffres binaires :
signe s :
codage = 0
valeur = 0
exposant e :
codage = 10000000001

```
 valeur = 1025
 mantisse m :
  codage = 0000000000010000011000100100110111010010111100011011
  valeur = 1125899906843

 Codage de          4.0001 = 4.0 + 10^(- 4) :
Codage 64 chiffres binaires :
 signe s :
  codage = 0
  valeur = 0
 exposant e :
  codage = 10000000001
  valeur = 1025
 mantisse m :
  codage = 0000000000000001101000110110111000101110101100011100
  valeur = 112589990684

 Codage de         4.00001 = 4.0 + 10^(- 5) :
Codage 64 chiffres binaires :
 signe s :
  codage = 0
  valeur = 0
 exposant e :
  codage = 10000000001
  valeur = 1025
 mantisse m :
  codage = 0000000000000000010100111110001011010110000100011100
  valeur = 11258999068

 Codage de        4.000001 = 4.0 + 10^(- 6) :
Codage 64 chiffres binaires :
 signe s :
  codage = 0
  valeur = 0
 exposant e :
  codage = 10000000001
  valeur = 1025
 mantisse m :
  codage = 0000000000000000000010000110001101111011110100000011
  valeur = 1125899907

 Codage de       4.0000001 = 4.0 + 10^(- 7) :
Codage 64 chiffres binaires :
 signe s :
  codage = 0
```

valeur = 0
exposant e :
codage = 10000000001
valeur = 1025
mantisse m :
codage = 0000000000000000000000000011010110101111111010100111
valeur = 112589991

Codage de 4.00000001 = 4.0 + 10^(- 8) :
Codage 64 chiffres binaires :
signe s :
codage = 0
valeur = 0
exposant e :
codage = 10000000001
valeur = 1025
mantisse m :
codage = 0000000000000000000000000010101011110011000111011 1
valeur = 11258999

Codage de 4.000000001 = 4.0 + 10^(- 9) :
Codage 64 chiffres binaires :
signe s :
codage = 0
valeur = 0
exposant e :
codage = 10000000001
valeur = 1025
mantisse m :
codage = 0000000000000000000000000001000100101110000011 00
valeur = 1125900

Codage de 4.0000000001 = 4.0 + 10^(-10) :
Codage 64 chiffres binaires :
signe s :
codage = 0
valeur = 0
exposant e :
codage = 10000000001
valeur = 1025
mantisse m :
codage = 0000000000000000000000000000011011011111001110
valeur = 112590

Codage de 4.00000000001 = 4.0 + 10^(-11) :

Codage 64 chiffres binaires :
 signe s :
  codage = 0
  valeur = 0
 exposant e :
  codage = 10000000001
  valeur = 1025
 mantisse m :
  codage = 0000000000000000000000000000000000000010101111111011
  valeur = 11259

 Codage de 4.000000000001 = 4.0 + 10^(-12) :
Codage 64 chiffres binaires :
 signe s :
  codage = 0
  valeur = 0
 exposant e :
  codage = 10000000001
  valeur = 1025
 mantisse m :
  codage = 0000000000000000000000000000000000000010001100110
  valeur = 1126

 Codage de 4.0000000000001 = 4.0 + 10^(-13) :
Codage 64 chiffres binaires :
 signe s :
  codage = 0
  valeur = 0
 exposant e :
  codage = 10000000001
  valeur = 1025
 mantisse m :
  codage = 0000000000000000000000000000000000000001110001
  valeur = 113

 Codage de 4.00000000000001 = 4.0 + 10^(-14) :
Codage 64 chiffres binaires :
 signe s :
  codage = 0
  valeur = 0
 exposant e :
  codage = 10000000001
  valeur = 1025
 mantisse m :
  codage = 0000000000000000000000000000000000000000001011

```
 valeur = 11

 Codage de 4.000000000000001 = 4.0 + 10^(-15) :
Codage 64 chiffres binaires :
 signe s :
  codage = 0
  valeur = 0
 exposant e :
  codage = 10000000001
  valeur = 1025
 mantisse m :
  codage = 0000000000000000000000000000000000000000000000000001
  valeur = 1

 Codage de               4 = 4.0 + 10^(-16) :
Codage 64 chiffres binaires :
 signe s :
  codage = 0
  valeur = 0
 exposant e :
  codage = 10000000001
  valeur = 1025
 mantisse m :
  codage = 0000000000000000000000000000000000000000000000000000
  valeur = 0


 --------
| PART 3 |
 --------

 Valeur de r1 = 0.9 - 0.6 - 0.3 (valeur attendue r1 = 0) :

    5.551D-17

 Mise en évidence des erreurs faites par l'ordinateur dans le calcul de 0.9-0.6-0.3 :

 Valeur de 0.3 :
 2.99999999999999988897769753748434595763683319091796875e-01

 Valeur de 0.6 :
 5.99999999999999977795539507496869191527366638183593750e-01

 Valeur de 0.9 :
 9.00000000000000022204460492503130808472633361816406250e-01
```

Valeur de 0.9-0.6 :
3.000000000000000444089209850062616169452667236328125e-01

Valeur de r2 = 0.9 - 0.3 - 0.6 (valeur attendue r2 = r1 = 0) :

1.110D-16

.....

Accumulation des erreurs dans les boucles :

ex 1 d'accumulation d'erreur dans une boucle : calcul de s = 0 + 0.3*100000 - 3*10000

Valeur de s = 0 + 0.3 * 100000 - 3 * 10000 (valeur attendue s = 0)

- 4.939D-08

ex 2 d'accumulation d'erreur dans une boucle : calcul de b/(2 ^ k)
La boucle 'while b > 0' s'est arretee au bout de 1075 itérations.
La boucle 'while b+1 > 1' s'est arretee au bout de 53 itérations.

.....
Calcul d'une approximation de pi ~ 3.141592653589793 :
n = 1, v(1) = 2.828427124746190
n = 2, v(2) = 3.061467458920719
n = 3, v(3) = 3.121445152258053
n = 4, v(4) = 3.136548490545941
n = 5, v(5) = 3.140331156954739
n = 6, v(6) = 3.141277250932757
n = 7, v(7) = 3.141513801144145
n = 8, v(8) = 3.141572940367883
n = 9, v(9) = 3.141587725279961
n = 10, v(10) = 3.141591421504635
n = 11, v(11) = 3.141592345611077
n = 12, v(12) = 3.141592576545004
n = 13, v(13) = 3.141592633463248
n = 14, v(14) = 3.141592654807589
n = 15, v(15) = 3.141592645321215
n = 16, v(16) = 3.141592607375720
n = 17, v(17) = 3.141592910939673
n = 18, v(18) = 3.141594125195191
n = 19, v(19) = 3.141596553704820
n = 20, v(20) = 3.141596553704820
n = 21, v(21) = 3.141674265021758

n = 22, v(22) = 3.141829681889202
n = 23, v(23) = 3.142451272494134
n = 24, v(24) = 3.142451272494134
n = 25, v(25) = 3.162277660168380
n = 26, v(26) = 3.162277660168380
n = 27, v(27) = 3.464101615137754
n = 28, v(28) = 4.000000000000000
n = 29, v(29) = 0.000000000000000
n = 30, v(30) = 0.000000000000000
Valeur attendue : pi ~ 3.141592653589793.

Calcul d'une deuxieme approximation de pi :
n = 1, v(1) = 2.828427124746190
n = 2, v(2) = 3.061467458920719
n = 3, v(3) = 3.121445152258053
n = 4, v(4) = 3.136548490545941
n = 5, v(5) = 3.140331156954739
n = 6, v(6) = 3.141277250932757
n = 7, v(7) = 3.141513801144145
n = 8, v(8) = 3.141572940367883
n = 9, v(9) = 3.141587725279961
n = 10, v(10) = 3.141591421504635
n = 11, v(11) = 3.141592345611077
n = 12, v(12) = 3.141592576545004
n = 13, v(13) = 3.141592633463248
n = 14, v(14) = 3.141592654807589
n = 15, v(15) = 3.141592645321215
n = 16, v(16) = 3.141592607375720
n = 17, v(17) = 3.141592910939673
n = 18, v(18) = 3.141594125195191
n = 19, v(19) = 3.141596553704820
n = 20, v(20) = 3.141596553704820
n = 21, v(21) = 3.141674265021758
n = 22, v(22) = 3.141829681889202
n = 23, v(23) = 3.142451272494134
n = 24, v(24) = 3.142451272494134
n = 25, v(25) = 3.162277660168380
n = 26, v(26) = 3.162277660168380
n = 27, v(27) = 3.464101615137754
n = 28, v(28) = 4.000000000000000
n = 29, v(29) = 0.000000000000000
n = 30, v(30) = 0.000000000000000
Valeur attendue : pi ~ 3.141592653589793.

Calcul de la deuxieme approximation (simplifiee) de pi :

n = 1, v(1) =   2.828427124746190
n = 2, v(2) =   3.061467458920718
n = 3, v(3) =   3.121445152258052
n = 4, v(4) =   3.136548490545939
n = 5, v(5) =   3.140331156954753
n = 6, v(6) =   3.141277250932773
n = 7, v(7) =   3.141513801144301
n = 8, v(8) =   3.141572940367092
n = 9, v(9) =   3.141587725277160
n = 10, v(10) =   3.141591421511200
n = 11, v(11) =   3.141592345570118
n = 12, v(12) =   3.141592576584873
n = 13, v(13) =   3.141592634338564
n = 14, v(14) =   3.141592648776986
n = 15, v(15) =   3.141592652386592
n = 16, v(16) =   3.141592653288993
n = 17, v(17) =   3.141592653514594
n = 18, v(18) =   3.141592653570994
n = 19, v(19) =   3.141592653585094
n = 20, v(20) =   3.141592653588619
n = 21, v(21) =   3.141592653589501
n = 22, v(22) =   3.141592653589721
n = 23, v(23) =   3.141592653589776
n = 24, v(24) =   3.141592653589790
n = 25, v(25) =   3.141592653589794
n = 26, v(26) =   3.141592653589794
n = 27, v(27) =   3.141592653589795
n = 28, v(28) =   3.141592653589795
n = 29, v(29) =   3.141592653589795
n = 30, v(30) =   3.141592653589795
Valeur attendue : pi ~ 3.141592653589793.