A taste of Julia

Αξιοσημείωτα Χαρακτηριστικά

1. Οικεία γραμμή εντολών

In [1]:
```
#Παράδειγμα δημιουργίας ενός δισδυάστατου πίνακα 4x4 που περιέχει αριθμούς κινητής υποδιαστολης με διπλάσια ακρίβεια.
using LinearAlgebra

A = rand(4,4) + 5*I
#I = eye(n,m) των Matlab, Octave, Python's Numpy etc.
```

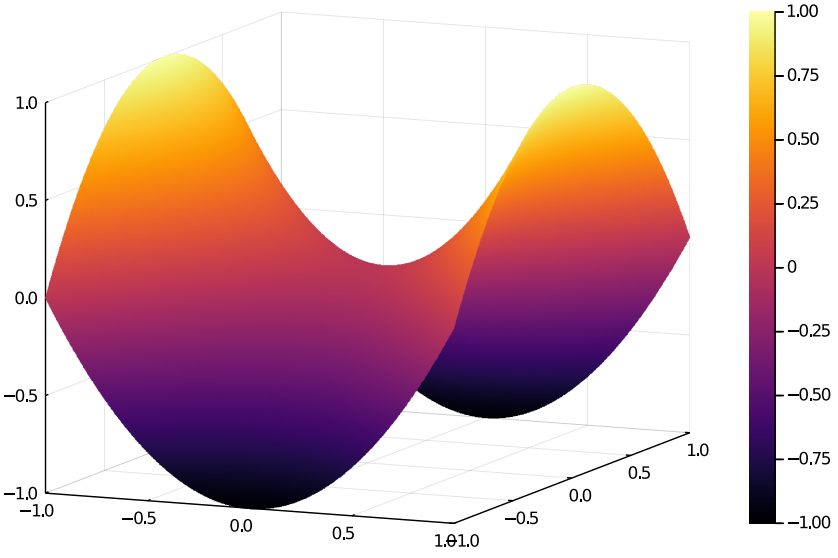
Out[1]:
```
4×4 Array{Float64,2}:
 5.82898   0.901527  0.646624  0.390918
 0.643695  5.13297   0.288222  0.551054
 0.966299  0.783438  5.76913   0.994315
 0.765362  0.039635  0.823325  5.83263
```

In [2]:
```
#Παραδείγματα αναφοράς σε στοιχεία πινάκων.

A = rand(2,2)
display(A)

x = A[1,2]
println("x = ", x)

y = (A + 2I)[2,2]
println("y = ", y)
#Σκεφτείτε το (A + 2I) ως έναν πίνακα B. Δεν χρειάζεται να κάνουμε ανάθεση σε μεταβλητή.
```

```
2×2 Array{Float64,2}:
 0.462245  0.699516
 0.570543  0.186287
x = 0.6995161422192246
y = 2.186287380581473
```

In [3]:
```
using Plots
x=-1:0.01:1
y=-1:0.01:1
h(x,y)=x^2-y^2;

surface(x,y,h)
```

Out[3]:

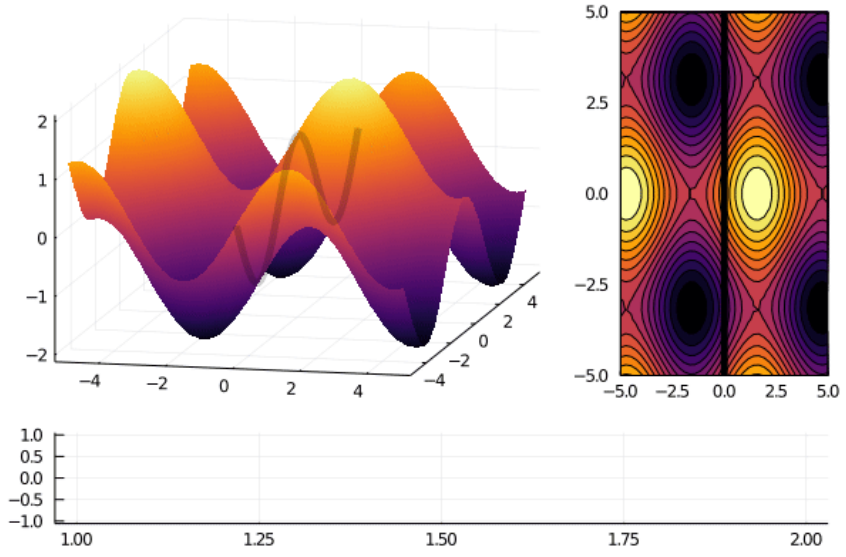


In [4]:

```
using Plots
default(legend = false)
x = y = range(-5, 5, length = 40)
zs = zeros(0, 40)
n = 100

@gif for i in range(0, stop = 2π, length = n)
    f(x, y) = sin(x + 10sin(i)) + cos(y)

    # create a plot with 3 subplots and a custom layout
    l = @layout [a{0.7w} b; c{0.2h}]
    p = plot(x, y, f, st = [:surface, :contourf], layout = l)

    # induce a slight oscillating camera angle sweep, in degrees (azimuth, altitude)
    plot!(p[1], camera = (10 * (1 + cos(i)), 40))

    # add a tracking line
    fixed_x = zeros(40)
    z = map(f, fixed_x, y)
    plot!(p[1], fixed_x, y, z, line = (:black, 5, 0.2))
    vline!(p[2], [0], line = (:black, 5))

    # add to and show the tracked values over time
    global zs = vcat(zs, z')
    plot!(p[3], zs, alpha = 0.2, palette = cgrad(:blues).colors)
end
```

```
┌ Info: Saved animation to
│   fn = C:\Users\Anastasia-Efterpi\tmp.gif
└ @ Plots C:\Users\Anastasia-Efterpi\.julia\packages\Plots\SjqWU\src\animation.jl:104
```

Out[4]:



2. Πληθώρα συναρτήσεων

Η Julia περιέχει πληθώρα συναρτήσεων που μειώνει σημαντικά το χρόνο και χώρο υλοποίησης αλγορίθμων.

In [5]:

```
#=
Παράδειγμα της συνάρτησης του τριδιαγώνιου πίνακα, όπου αποθηκεύονται στη μνήμη μόνο τα απαραίτητα στοιχεία του-όχι τα μηδενικά.

SymTridiagonal είναι η συνάρτηση του συμμετρικού τριδιαγώνιου πίνακα. (Υπάρχει ενσωματωμένη στη Julia)
strang(n) είναι δική μας συνάρτηση, τη δημιουργήσαμε εμείς ως χρήστες.
Την ονομάσαμε strang, λόγω του ότι ο συγκεκριμένος πίνακας είναι ο αγαπημένος του μαθηματικού Gil Strang.
=#

strang(n) = SymTridiagonal(2*ones(n), -ones(n-1))
strang(7)
```

Out[5]:

```
7×7 SymTridiagonal{Float64,Array{Float64,1}}:
  2.0  -1.0    ·     ·     ·     ·     ·
 -1.0   2.0  -1.0    ·     ·     ·     ·
   ·   -1.0   2.0  -1.0    ·     ·     ·
   ·     ·   -1.0   2.0  -1.0    ·     ·
   ·     ·     ·   -1.0   2.0  -1.0    ·
   ·     ·     ·     ·   -1.0   2.0  -1.0
   ·     ·     ·     ·     ·   -1.0   2.0
```

In [11]:
```
#=
Το γεγονός ότι αποθηκεύονται στη μνήμη τα απαραίτητα στοιχεία του πίνακα, μειώνει δραματικά το χρόνο επίλυσης της εξίσωσης Ax=b.
Χρόνος O(n).
Ax = b, στην Julia συμβολίζεται ως A\b
=#

strang(n) = SymTridiagonal(2*ones(n), -ones(n-1))
n = 1000
A = strang(n)
#Το μέγεθος ενός πίνακα nxn.
@show n*n
#Το μέγεθος του πίνακα μας μέσω της χρήσης της συνάρτησης SymTridiagonal.
@show sizeof(A)

#Η σημαντική διαφορά στους χρόνους
b = ones(n)
@time A\b

C = Matrix(A)

@time C\b
print()
```

```
n * n = 1000000
sizeof(A) = 16
  0.000017 seconds (3 allocations: 23.812 KiB)
  0.096033 seconds (4 allocations: 7.645 MiB)
```

3. Multiple Dispatching

Η Julia μπορεί να αναγνωρίσει τις εισόδους των συναρτήσεων και να επιλέξει ποια συνάρτηση να χρησιμοποιήσει από μια πληθώρα μεθόδων, καθώς επίσης μπορούμε να ορίσουμε και δικές μας συναρτήσεις με συγκεκριμένα ορίσματα.

In [7]:
```
#Πρόσθεση ακεραίων.
add(x::Int, y::Int) = x+y

#Πρόσθεση αριθμών κινητής υποδιαστολής.
vaddsd(x::Float64, y::Float64) = x+y

@show a = add(2,3)
@show b = vaddsd(2.5, 2.5)

#Αλλά έχουμε error αν:
c = add(2.5, 3)
```

```
a = add(2, 3) = 5
b = vaddsd(2.5, 2.5) = 5.0
MethodError: no method matching add(::Float64, ::Int64)
Closest candidates are:
  add(::Int64, ::Int64) at In[7]:2
```

In [8]:
```
#Αθροίσματα πινάκων με σύμβολο \oplus TAB.
import SparseArrays: SparseMatrixCSC
using SparseArrays

#Πυκνός + Πυκνός
⊕(A::Matrix, B::Matrix) =
    [A[i,j]+B[i,j] for i in 1:size(A,1), j in 1:size(A,2)]

#Πυκνός + Αραιός
⊕(A::Matrix, B::AbstractSparseMatrix) = A ⊕ full(B)

#Αραιός + Πυκνός
⊕(A::AbstractSparseMatrix, B::Matrix) = B ⊕ A
```

Out[8]: ⊕ (generic function with 3 methods)

In [12]:
```
#Συνάρτηση υπολογισμού ορίζουσας.

using LinearAlgebra
#Αριθμού
newdet(x::Number) = x

#Διαγώνιου
newdet(A::Diagonal) = prod(diag(A))

#Τριδιαγώνιου
newdet(A::UpperTriangular) = prod(diag(A))
```

```
#Πίνακα
newdet(A::Matrix) = -prod(diag(qr(Matrix(A)).R))*(-1)^size(A,1)

n = 500
A = Diagonal(diagm(rand(n)))
B = UpperTriangular(triu(rand(n,n)))
C = rand(n,n)

@time newdet(A)
@time newdet(B)
@time newdet(C)
print()
```

```
  0.007339 seconds (3.39 k allocations: 163.924 KiB)
  0.006984 seconds (3.46 k allocations: 166.949 KiB)
  0.021769 seconds (24.54 k allocations: 7.203 MiB)
```

4. Κώδικας μηχανής

Μπορούμε να δούμε τον κώδικα μηχανής μέσω του `@code_native` χωρίς να χρειαστεί να ακολουθήσουμε κάποια ιδιαίτερη διαδικασία, όπως στις άλλες γλώσσες προγραμματισμού.

In [10]:

```
f(a,b) = a + b
@code_native f(1.0,2.0)

#Στον κώδικα assembly η εντολή vaddsd εκτελεί την πρόσθεση.
```

```
        .text
; ┌ @ In[10]:1 within `f'
        pushq   %rbp
        movq    %rsp, %rbp
; │┌ @ float.jl:401 within `+'
        vaddsd  %xmm1, %xmm0, %xmm0
; │└
        popq    %rbp
        retq
        nopw    (%rax,%rax)
; └
```