MTH8408 : Méthodes d'optimisation et contrôle optimal

Laboratoire 5: Optimisation avec contraintes et calcul variationnel

Ulrich Baron-Fournier (2021196)

Tangi Migot et Paul Raynaud

In [49]:
```
using Krylov, LinearAlgebra, Logging, NLPModels, NLPModelsIpopt, Printf, SolverCore
```

In [50]:
```
using PDENLPModels, Gridap, PlotlyJS
```

Quelques commentaires en Julia

Les kwargs: choix optionnels

Dans le projet du dernier labo, une des questions demandait d'ajouter une option pour utiliser la fonction `lsmr` ou `lsqr`. C'est le cas typique d'arguments optionnels:

- On veut proposer un choix par défaut à l'utilisateur, par exemple `lsqr` ;
- On veut laisser la possibilité à l'utilisateur de changer;
- On voudrait aussi pouvoir ajouter d'autres par la suite (sans avoir à tout modifier).

In [51]:
```
function dsol(A, b, ϵ; solver :: Function = lsqr)
    (d, stats) = solver(A, b, atol = ϵ)
    return d
end
```

```
dsol (generic function with 1 method)
```

A noter que l'on donne des valeurs par défaut aux arguments qui apparaissent après le `;`.

Exercice 1: Pénalité quadratique pour les ADNLPModels

Dans cet exercice, on va étudier une version simple d'une méthode de pénalité quadratique pour les problèmes d'optimisation avec contraintes d'égalité.

```
math
min f(x) s.à c(x) = 0.
```

Dans les labos précédents, on a déjà utilisé un NLPModel particulier, le ADNLPModel:

In [52]:
```
using ADNLPModels, LinearAlgebra, Test
fH(x) = (x[2]+x[1].^2-11)^2 + (x[1]+x[2].^2-7)^2
x0H = [10., 20.]
cH(x) = [x[1]-1]
himmelblau = ADNLPModel(fH, x0H, cH, [0.], [0.])
```

```
ADNLPModel - Model with automatic differentiation backend ADModelBackend{
  ForwardDiffADGradient,
  ForwardDiffADHvprod,
  ForwardDiffADJprod,
  ForwardDiffADJtprod,
  ForwardDiffADJacobian,
  ForwardDiffADHessian,
  ForwardDiffADGHjvprod,
}
  Problem name: Generic
   All variables: ████████████████████ 2      All constraints: ████████████████████
1
            free: ████████████████████ 2                free: ····················· 0
           lower: ····················· 0               lower: ····················· 0
           upper: ····················· 0               upper: ····················· 0
         low/upp: ····················· 0             low/upp: ····················· 0
           fixed: ····················· 0               fixed: ████████████████████ 1
          infeas: ····················· 0              infeas: ····················· 0
            nnzh: (  0.00% sparsity)   3                       linear: ····················· 0
                                                          nonlinear: ████████████████████
1
                                                               nnzj: (  0.00% sparsity)
2

  Counters:
             obj: ····················· 0                 grad: ····················· 0
cons: ····················· 0
        cons_lin: ····················· 0             cons_nln: ····················· 0
jcon: ····················· 0
           jgrad: ····················· 0                  jac: ····················· 0              ja
c_lin: ····················· 0
         jac_nln: ····················· 0                jprod: ····················· 0            jpro
d_lin: ····················· 0
       jprod_nln: ····················· 0              jtprod: ····················· 0           jtpro
d_lin: ····················· 0
      jtprod_nln: ····················· 0                 hess: ····················· 0
hprod: ····················· 0
           jhess: ····················· 0              jhprod: ····················· 0
```

Attention: dans toute la suite de l'exercice on suppose que les bornes sur les contraintes `nlp.meta.lcon` et `nlp.meta.ucon` sont 0 pour simplifier.

Question 1: Transformer un ADNLPModel en un problème pénalisé

Coder la fonction `quad_penalty_adnlp` qui prend en entrée un ADNLPModel, et un paramètre ρ et qui retourne un nouveau ADNLPModel qui correspond au problème sans contrainte: $$ \min_x f(x) + \frac{\rho}{2}\|c(x)\|^2. $$ Remarque: on peut accèder aux fonctions f et c par `NLPModels.obj()` et `NLPModels.cons()`.

In [53]:
```
function quad_penalty_adnlp(nlp :: ADNLPModel, ρ :: Real)
    f = x -> NLPModels.obj(nlp, x) + ρ/2 * norm(NLPModels.cons(nlp, x))^2
    nlp_quad = ADNLPModel(f, nlp.meta.x0)
    return nlp_quad
end
```

```
quad_penalty_adnlp (generic function with 1 method)
```

In [54]:
```
#Faire des tests pour vérifier que ça fonctionne.
# fH(x) = (x[2]+x[1].^2-11).^2+(x[1]+x[2].^2-7).^2
# x0H = [10., 20.]
# himmelblau = ADNLPModel(fH, x0H)

himmelblau_quad = quad_penalty_adnlp(himmelblau, 1)
@test himmelblau_quad.meta.ncon == 0
@test obj(himmelblau_quad, zeros(2)) == 170.5
```

```
Test Passed
```

In [55]:
```
#Ajouter au moins un autre test similaire avec des contraintes.
@test himmelblau.meta.ncon == 1
@test obj(himmelblau, zeros(2)) == 170.5-1/2*norm(NLPModels.cons(himmelblau, zeros(
```

```
Test Passed
```

In [56]:
```
# Ajouter un test au cas ou `nlp.meta.lcon` ou `nlp.meta.ucon` ont des composantes
@test himmelblau.meta.lcon == [0.]
@test himmelblau.meta.ucon == [0.]
```

```
Test Passed
```

Question 2: KKT

Coder une fonction `KKT_eq_constraint(nlp :: AbstractNLPModel, x, λ)` qui vérifie si le point `x` avec multiplicateur de Lagrange `λ` satisfait les conditions KKT d'un problème avec contraintes d'égalités.

In [57]:
```
function KKT_eq_constraint(nlp :: AbstractNLPModel, x, λ)
   ∇f = grad(nlp, x)
   J = NLPModels.jac(nlp,x)
   satisfyKKT =  isapprox(∇f, J'λ, atol = 1e-10)
   satisfyKKT_cons = isapprox(NLPModels.cons(nlp, x), zeros(size(λ)), atol = 1e-10)
   satisfy = satisfyKKT && satisfyKKT_cons
   return satisfy
end
```

```
KKT_eq_constraint (generic function with 1 method)
```

In [58]:
```
#test
fH(x) = x[1] + x[2]
```

```
x0H = [0., 1.]
cH(x) = x[1]^2 + x[2]^2 - 1
himmelblau = ADNLPModel(fH, x0H, cH, [0.], [0.])
satisfy = KKT_eq_constraint(himmelblau, [1/sqrt(2), 1/sqrt(2)], 1/sqrt(2))
@test satisfy == true
```

```
Test Passed
```

Question 3: méthode de pénalité quadratique

In [59]:
```
using NLPModelsIpopt
```

In [60]:
```
function quad_penalty(nlp      :: AbstractNLPModel,
                      x        :: AbstractVector;
                      ϵ        :: AbstractFloat = 1e-3,
                      η        :: AbstractFloat = 1e6,
                      σ        :: AbstractFloat = 2.0,
                      max_eval :: Int = 1_000,
                      max_time :: AbstractFloat = 60.,
                      max_iter :: Int = typemax(Int64),
                      ρ        :: AbstractFloat = 1.0
                      )
    ##### Initialiser cx et gx au point x;
    cx = cons(nlp, x) # Initialiser la violation des contraintes
    gx = grad(nlp, x) # Initialiser le gradient
    ########################################################
    normcx = normcx_old = norm(cx)


    iter = 0

    el_time = 0.0
    tired   = neval_cons(nlp) > max_eval || el_time > max_time
    status  = :unknown

    start_time = time()
    too_small  = false
    normdual   = norm(gx) #exceptionnellement on ne va pas vérifier toute l'optimal
    optimal    = max(normcx, normdual) ≤ ϵ

    nlp_quad   = quad_penalty_adnlp(nlp, ρ)

    @info log_header([:iter, :nf, :primal, :status, :nd, :Δ],
    [Int, Int, Float64, String, Float64, Float64],
    hdr_override=Dict(:nf => "#F", :primal => "‖F(x)‖", :nd => "‖d‖"))

    while !(optimal || tired || too_small)

        #Appeler Ipopt pour résoudre le problème pénalisé en partant du point x0 =
        #utiliser l'option print_level = 0 pour enlever les affichages d'ipopt.
        stats = ipopt(nlp_quad, print_level = 0)
        #...
        ################################################

        if stats.status == :first_order
```

```
            ###### Mettre à jour cx avec la solution renvoyé par Ipopt
            x = stats.solution
            #...
            cx = cons(nlp, x)
            #...
            ################################################################
            normcx_old = normcx
            normcx = norm(cx)
        end

        if normcx_old > 0.95 * normcx
            ρ *= σ
        end

        @info log_row(Any[iter, neval_cons(nlp), normcx, stats.status])

        nlp_quad   = quad_penalty_adnlp(nlp, ρ)

        el_time      = time() - start_time
        iter   += 1
        many_evals   = neval_cons(nlp) > max_eval
        iter_limit   = iter > max_iter
        tired        = many_evals || el_time > max_time || iter_limit || ρ ≥ η
        ##### Utiliser la réalisabilité dual renvoyé par Ipopt pour normdual
        normdual     =  stats.dual_feas
        #....
        ########################################################################
        optimal      = max(normcx, normdual) ≤ ϵ
    end

    status = if optimal
        :first_order
    elseif tired
        if neval_cons(nlp) > max_eval
            :max_eval
        elseif el_time > max_time
            :max_time
        elseif iter > max_iter
            :max_iter
        else
            :unknown_tired
        end
    elseif too_small
        :stalled
    else
        :unknown
    end

    return GenericExecutionStats(nlp, status = status, solution = x,
                                 objective = obj(nlp, x),
                                 primal_feas = normcx,
                                 dual_feas = normdual,
                                 iter = iter,
                                 elapsed_time = el_time,
                                 solver_specific = Dict(:penalty => ρ))
end
```

```
quad_penalty (generic function with 1 method)
```

In [61]:
```
#Faire des tests pour vérifier que ça fonctionne.
stats = quad_penalty(himmelblau, x0H)
@test stats.status == :first_order
@test stats.solution ≈ [1.0008083416169895, 2.709969135758311] atol=1e-2
@test norm(cons(himmelblau, stats.solution)) ≈ 0. atol=1e-3
```

```
┌ Info:   iter      #F    ‖F(x)‖           status       ‖d‖         Δ
└ @ Main c:\Users\Ulrizpascuit\Desktop\MTH8408\MTH8408-Hiv24\lab5\Lab5-notebook.ipyn
b:31
┌ Info:      0      47   5.7e-01      first_order
└ @ Main c:\Users\Ulrizpascuit\Desktop\MTH8408\MTH8408-Hiv24\lab5\Lab5-notebook.ipyn
b:58
┌ Info:      1      92   5.7e-01      first_order
└ @ Main c:\Users\Ulrizpascuit\Desktop\MTH8408\MTH8408-Hiv24\lab5\Lab5-notebook.ipyn
b:58
┌ Info:      2     141   3.1e-01      first_order
└ @ Main c:\Users\Ulrizpascuit\Desktop\MTH8408\MTH8408-Hiv24\lab5\Lab5-notebook.ipyn
b:58
┌ Info:      3     193   1.6e-01      first_order
└ @ Main c:\Users\Ulrizpascuit\Desktop\MTH8408\MTH8408-Hiv24\lab5\Lab5-notebook.ipyn
b:58
┌ Info:      4     256   8.5e-02      first_order
└ @ Main c:\Users\Ulrizpascuit\Desktop\MTH8408\MTH8408-Hiv24\lab5\Lab5-notebook.ipyn
b:58
┌ Info:      5     318   4.3e-02      first_order
└ @ Main c:\Users\Ulrizpascuit\Desktop\MTH8408\MTH8408-Hiv24\lab5\Lab5-notebook.ipyn
b:58
┌ Info:      6     391   2.2e-02      first_order
└ @ Main c:\Users\Ulrizpascuit\Desktop\MTH8408\MTH8408-Hiv24\lab5\Lab5-notebook.ipyn
b:58
┌ Info:      7     482   1.1e-02      first_order
└ @ Main c:\Users\Ulrizpascuit\Desktop\MTH8408\MTH8408-Hiv24\lab5\Lab5-notebook.ipyn
b:58
┌ Info:      8     588   5.5e-03      first_order
└ @ Main c:\Users\Ulrizpascuit\Desktop\MTH8408\MTH8408-Hiv24\lab5\Lab5-notebook.ipyn
b:58
┌ Info:      9     711   2.8e-03      first_order
└ @ Main c:\Users\Ulrizpascuit\Desktop\MTH8408\MTH8408-Hiv24\lab5\Lab5-notebook.ipyn
b:58
┌ Info:     10     896   1.4e-03      first_order
└ @ Main c:\Users\Ulrizpascuit\Desktop\MTH8408\MTH8408-Hiv24\lab5\Lab5-notebook.ipyn
b:58
┌ Info:     11    1079   6.9e-04      first_order
└ @ Main c:\Users\Ulrizpascuit\Desktop\MTH8408\MTH8408-Hiv24\lab5\Lab5-notebook.ipyn
b:58
```

Test Failed at **c:\Users\Ulrizpascuit\Desktop\MTH8408\MTH8408-Hiv24\lab5\Lab5-notebook.ipynb:4**

```
  Expression: ≈(stats.solution, [1.0008083416169895, 2.709969135758311], atol = 0.0
1)
   Evaluated: [-0.7073507954872859, -0.7073507954872853] ≈ [1.0008083416169895, 2.70
9969135758311] (atol=0.01)
```

```
Test.FallbackTestSetException("There was an error during testing")



Stacktrace:

 [1] record(ts::Test.FallbackTestSet, t::Union{Test.Error, Test.Fail})

   @ Test C:\Users\Ulrizpascuit\AppData\Local\Programs\Julia-1.9.3\share\julia\stdli
b\v1.9\Test\src\Test.jl:960

 [2] do_test(result::Test.ExecutionResult, orig_expr::Any)

   @ Test C:\Users\Ulrizpascuit\AppData\Local\Programs\Julia-1.9.3\share\julia\stdli
b\v1.9\Test\src\Test.jl:670

 [3] top-level scope

   @ C:\Users\Ulrizpascuit\AppData\Local\Programs\Julia-1.9.3\share\julia\stdlib\v1.
9\Test\src\Test.jl:478
```

Vérifier que la solution rendue vérifie les conditions KKT avec la fonction de la question précédente.

In [62]:
```
satisfy = KKT_eq_constraint(himmelblau, stats.solution, 1/sqrt(2))
@test satisfy == true
```

```
Test Failed at c:\Users\Ulrizpascuit\Desktop\MTH8408\MTH8408-Hiv24\lab5\Lab5-noteboo
k.ipynb:2
  Expression: satisfy == true
   Evaluated: false == true
```

```
Test.FallbackTestSetException("There was an error during testing")



Stacktrace:

 [1] record(ts::Test.FallbackTestSet, t::Union{Test.Error, Test.Fail})

   @ Test C:\Users\Ulrizpascuit\AppData\Local\Programs\Julia-1.9.3\share\julia\stdli
b\v1.9\Test\src\Test.jl:960

 [2] do_test(result::Test.ExecutionResult, orig_expr::Any)

   @ Test C:\Users\Ulrizpascuit\AppData\Local\Programs\Julia-1.9.3\share\julia\stdli
b\v1.9\Test\src\Test.jl:670

 [3] top-level scope

   @ C:\Users\Ulrizpascuit\AppData\Local\Programs\Julia-1.9.3\share\julia\stdlib\v1.
9\Test\src\Test.jl:478
```

Exercice 2: Calcul Variationnel

Dans cet exercice, on considère le problème de calcul variationnel suivant: $$ \min \int_0^1 (\dot{x}(t)^2+2x(t)^2)e^t dt, \quad x(0)=0, x(1)=e - e^{-2} $$

modélisé avec `PDENLPModels`.

In [63]:
```
function cv_model(n :: Int)

  domain = (0,1) # set the domain
  partition = n
  model = CartesianDiscreteModel(domain,partition) # set discretization

  labels = get_face_labeling(model)
  add_tag_from_tags!(labels,"diri1",[2])
  add_tag_from_tags!(labels,"diri0",[1]) # boundary conditions

  order=1
  valuetype=Float64
  reffe = ReferenceFE(lagrangian, valuetype, order)
  V0 = TestFESpace(model, reffe; conformity=:H1, dirichlet_tags=["diri0","diri1"])
  U = TrialFESpace(V0,[0., exp(1)-exp(-2)])

  trian = Triangulation(model)
  degree = 2
  dΩ = Measure(trian,degree) # integration machinery

  # Our objective function
  w(x) = exp(x[1])
  function f(y)
    ∫((∇(y)⊙∇(y) + 2 * y * y) * w) * dΩ
  end

  xin = zeros(Gridap.FESpaces.num_free_dofs(U))
  nlp = GridapPDENLPModel(xin, f, trian, U, V0)
  return nlp
end
```

```
cv_model (generic function with 1 method)
```

Question 1: Résoudre

Résoudre le NLPModel généré par la fonction `cv_model` pour `n = 16` avec `ipopt` et afficher la solution (attention la solution rendue ne contient pas les valeurs aux bords qu'il faut rajouter).

In [64]:
```
n = 16
nlp = cv_model(n)
```

```
GridapPDENLPModel
  Problem name: Generic
   All variables: ████████████████ 15     All constraints: ···················· 0
                 free: ████████████████ 15                 free: ···················· 0
                lower: ···················· 0                lower: ···················· 0
                upper: ···················· 0                upper: ···················· 0
              low/upp: ···················· 0              low/upp: ···················· 0
                fixed: ···················· 0                fixed: ···················· 0
               infeas: ···················· 0               infeas: ···················· 0
                 nnzh: ( 63.33% sparsity)   44                linear: ···················· 0
                                                          nonlinear: ···················· 0
                                                               nnzj: (------% sparsity)

  Counters:
                 obj: ···················· 0                grad: ···················· 0
cons: ···················· 0
           cons_lin: ···················· 0             cons_nln: ···················· 0
jcon: ···················· 0
             jgrad: ···················· 0                 jac: ···················· 0              ja
c_lin: ···················· 0
            jac_nln: ···················· 0               jprod: ···················· 0            jpro
d_lin: ···················· 0
           jprod_nln: ···················· 0             jtprod: ···················· 0           jtpro
d_lin: ···················· 0
          jtprod_nln: ···················· 0               hess: ···················· 0
hprod: ···················· 0
             jhess: ···················· 0              jhprod: ···················· 0
```

In [65]:
```
stats = ipopt(nlp, print_level = 0)
a = 0.0
b = exp(1) - exp(1)^(-2)

x = range(0, stop=1, length=16+1)
sol = [a; stats.solution; b]

println(sol)
```

```
[0.0, 0.18196306770012807, 0.3542814202913607, 0.5188461331646763, 0.6773749095244441, 0.8314356102360283, 0.9824672217813848, 1.131798576544907, 1.2806651035693914, 1.4302238561349263, 1.5815670345204877, 1.7357341976668708, 1.8937233357803989, 2.056500956855748, 2.2250113233500324, 2.400184960541827, 2.5829465452224323]
```

In [66]:
```
trace = scatter(x=x, y=sol, mode="lines", name="n=16")
PlotlyJS.plot([trace], Layout(title="Solution", xaxis_title="x", yaxis_title="y"))
```

Question 2: Convergence en `n`

Afficher sur un même graphique la solution obtenue par `ipopt` pour plusieurs valeurs de `n`.

In [67]:
```
a = 0.0
b = exp(1) - exp(1)^(-2)
x_16 = range(0, stop=1, length=16+1)
x_64 = range(0, stop=1, length=64+1)
```

```
x_128 = range(0, stop=1, length=128+1)

nlp_16 = cv_model(16)
nlp_64 = cv_model(64)
nlp_128 = cv_model(128)

stats_16 = ipopt(nlp_16, print_level = 0)
stats_64 = ipopt(nlp_64, print_level = 0)
stats_128 = ipopt(nlp_128, print_level = 0)

y_16 = [a; stats_16.solution; b]
y_64 = [a; stats_64.solution; b]
y_128 = [a; stats_128.solution; b]

f = x -> exp(x) - exp(-2*x)
y = [f((i-1)/128) for i in 1:129]
```

```
129-element Vector{Float64}:
 0.0
 0.023346660201039593
 0.0465144741103416
 0.06950765063316955
 0.09233034468562695
 0.11498665812002529
 0.1374806406365986
 0.1598162906817815
 0.18199755633326387
 0.2040283361720343
 ⋮
 2.4002344912179985
 2.422640089951961
 2.445165910160412
 2.46781375532096
 2.490585429714178
 2.5134827386367493
 2.536507488613078
 2.5596614876054
 2.5829465452224323
```

In [68]:
```
trace16 = scatter(x=x_16, y=y_16, mode="lines", name="n=16")
trace64 = scatter(x=x_64, y=y_64, mode="lines", name="n=64")
trace128 = scatter(x=x_128, y=y_128, mode="lines", name="n=128")
trace = scatter(x=x_128, y=y, mode="lines", name="exact")
PlotlyJS.plot([trace16, trace64, trace128, trace], Layout(title="Multiple Lines Plo
```

Question 3: Comparer à la solution exacte

La solution exacte est $x(t)=e^t - e^{-2t}$ et la valeur optimale est $e^3 - 2e^{-3}+1$.

In [69]:
```
@test norm(stats_16.objective - (exp(3) - 2*exp(-3) + 1)) < 5e-5
```

```
Test Failed at c:\Users\Ulrizpascuit\Desktop\MTH8408\MTH8408-Hiv24\lab5\Lab5-notebook.ipynb:1
  Expression: norm(stats_16.objective - ((exp(3) - 2 * exp(-3)) + 1)) < 5.0e-5
   Evaluated: 0.0011116893401030836 < 5.0e-5
```

```
Test.FallbackTestSetException("There was an error during testing")



Stacktrace:

 [1] record(ts::Test.FallbackTestSet, t::Union{Test.Error, Test.Fail})

   @ Test C:\Users\Ulrizpascuit\AppData\Local\Programs\Julia-1.9.3\share\julia\stdlib\v1.9\Test\src\Test.jl:960

 [2] do_test(result::Test.ExecutionResult, orig_expr::Any)

   @ Test C:\Users\Ulrizpascuit\AppData\Local\Programs\Julia-1.9.3\share\julia\stdlib\v1.9\Test\src\Test.jl:670

 [3] top-level scope

   @ C:\Users\Ulrizpascuit\AppData\Local\Programs\Julia-1.9.3\share\julia\stdlib\v1.9\Test\src\Test.jl:478
```

In [70]:
```
@test norm(stats_64.objective - (exp(3) - 2*exp(-3) + 1)) < 5e-5
```

```
Test Failed at c:\Users\Ulrizpascuit\Desktop\MTH8408\MTH8408-Hiv24\lab5\Lab5-notebook.ipynb:1
  Expression: norm(stats_64.objective - ((exp(3) - 2 * exp(-3)) + 1)) < 5.0e-5
   Evaluated: 6.975787896479346e-5 < 5.0e-5
```

```
Test.FallbackTestSetException("There was an error during testing")



Stacktrace:

 [1] record(ts::Test.FallbackTestSet, t::Union{Test.Error, Test.Fail})

   @ Test C:\Users\Ulrizpascuit\AppData\Local\Programs\Julia-1.9.3\share\julia\stdlib\v1.9\Test\src\Test.jl:960

 [2] do_test(result::Test.ExecutionResult, orig_expr::Any)

   @ Test C:\Users\Ulrizpascuit\AppData\Local\Programs\Julia-1.9.3\share\julia\stdlib\v1.9\Test\src\Test.jl:670

 [3] top-level scope

   @ C:\Users\Ulrizpascuit\AppData\Local\Programs\Julia-1.9.3\share\julia\stdlib\v1.9\Test\src\Test.jl:478
```

In [71]:
```
@test norm(stats_128.objective - (exp(3) - 2*exp(-3) + 1)) < 5e-5
```

```
Test Passed
```

In [72]:
```
sol_exact = [f((i-1)/16) for i in 1:17]
@test norm(y_16 - sol_exact) < 5e-5
```

```
Test Failed at c:\Users\Ulrizpascuit\Desktop\MTH8408\MTH8408-Hiv24\lab5\Lab5-notebook.ipynb:2
  Expression: norm(y_16 - sol_exact) < 5.0e-5
   Evaluated: 0.0005191645760710027 < 5.0e-5
```

```
Test.FallbackTestSetException("There was an error during testing")



Stacktrace:

 [1] record(ts::Test.FallbackTestSet, t::Union{Test.Error, Test.Fail})

   @ Test C:\Users\Ulrizpascuit\AppData\Local\Programs\Julia-1.9.3\share\julia\stdlib\v1.9\Test\src\Test.jl:960

 [2] do_test(result::Test.ExecutionResult, orig_expr::Any)

   @ Test C:\Users\Ulrizpascuit\AppData\Local\Programs\Julia-1.9.3\share\julia\stdlib\v1.9\Test\src\Test.jl:670

 [3] top-level scope

   @ C:\Users\Ulrizpascuit\AppData\Local\Programs\Julia-1.9.3\share\julia\stdlib\v1.9\Test\src\Test.jl:478
```

In [73]:
```
sol_exact = [f((i-1)/64) for i in 1:65]
@test norm(y_64 - sol_exact) < 5e-5
```

```
Test Failed at c:\Users\Ulrizpascuit\Desktop\MTH8408\MTH8408-Hiv24\lab5\Lab5-notebook.ipynb:2
  Expression: norm(y_64 - sol_exact) < 5.0e-5
   Evaluated: 6.483407171232379e-5 < 5.0e-5
```

```
Test.FallbackTestSetException("There was an error during testing")



Stacktrace:

 [1] record(ts::Test.FallbackTestSet, t::Union{Test.Error, Test.Fail})

   @ Test C:\Users\Ulrizpascuit\AppData\Local\Programs\Julia-1.9.3\share\julia\stdli
b\v1.9\Test\src\Test.jl:960

 [2] do_test(result::Test.ExecutionResult, orig_expr::Any)

   @ Test C:\Users\Ulrizpascuit\AppData\Local\Programs\Julia-1.9.3\share\julia\stdli
b\v1.9\Test\src\Test.jl:670

 [3] top-level scope

   @ C:\Users\Ulrizpascuit\AppData\Local\Programs\Julia-1.9.3\share\julia\stdlib\v1.
9\Test\src\Test.jl:478
```

In [74]:
```
sol_exact = [f((i-1)/128) for i in 1:129]
@test norm(y_128 - sol_exact) < 5e-5
```

Test Passed