Name: Hongda Li
AMATH 585 2022 WINTER HW7

Problem 1

The conjugate gradient algorithm is given as:

$$\begin{aligned}
&p_0 = b - Ax_0 \\
&\text{For } i = 0, 1, \cdots \\
&\quad a_i = \frac{\|r_i\|^2}{\|p_i\|_A^2} \\
&\quad x_{i+1} = x_i + a_i p_i \\
&\quad r_{i+1} = r_i - a_i A p_i \\
&\quad b_i = \frac{\|r_{i+1}\|_2^2}{\|r_i\|_2^2} \\
&\quad p_{i+1} = r_{i+1} + b_i p_i
\end{aligned}$$

The algorithm has been rephrased. And we make the assumption that the matrix A is symmetric poisitive definite. In addition, observe that a_i, b_i are non-negative real numbers. This means that we can move then around even if the vector in the inner products can be complex. We wish to prove 3 hypothesis about the algorithm inductively:

$$\begin{aligned}
\mathcal{H}_1(k) &\equiv \forall\, 0 \le j \le k-1 : \langle r_k, p_j\rangle = 0 \\
\mathcal{H}_2(k) &\equiv \forall\, 0 \le j \le k-1 : \langle p_k, Ap_j\rangle = 0 \\
\mathcal{H}_3(k) &\equiv \forall\, 0 \le j \le k-1 : \langle r_k, r_j\rangle = 0
\end{aligned} \tag{1.1}$$

First we verify the basecase by considering: $\mathcal{H}_1(1), \mathcal{H}_2(1), \mathcal{H}_3(1)$.

$$\begin{aligned}
\langle r_1, r_0\rangle &= \langle r_0 - a_0 A p_0, r_0\rangle \\
&= \langle r_0, p_0\rangle - a_0\langle r_0, Ap_0\rangle \\
&= \langle r_0, r_0\rangle - a_0\langle r_0, Ar_0\rangle \\
&= 0 \\
&\implies \mathcal{H}_3(1) \text{ is true} \\
\langle p_1, Ap_0\rangle &= \langle r_1, Ap_0\rangle + \frac{\langle r_1, r_1\rangle}{\langle r_0, r_0\rangle}\langle p_0, Ap_0\rangle \\
&= \langle r_1, a_0^{-1}(r_0 - r_1)\rangle + a_0^{-1}\langle r_1, r_1\rangle \\
\mathcal{H}_3(1) \implies &= -a_0^{-1}\langle r_1, r_1\rangle + a_0^{-1}\langle r_1, r_1\rangle \\
&\implies \mathcal{H}_2(1) \text{ is true} \\
\langle r_1, p_0\rangle &= \langle r_1, r_0\rangle \\
\mathcal{H}_3(1) \implies &= 0 \\
&\implies \mathcal{H}_1(1) \text{ is true}
\end{aligned} \tag{1.2}$$

Basecase is asserted by the definition of the starting conditions and the coefficient a_0. next we assume that $\mathcal{H}_1(k), \mathcal{H}(k), \mathcal{H}(k)$ are all true, and then we wish to prove inductively that they remainds to be true. First, we establish some equalities that are not obvious to simplify the proof, and then we prove it.

$$\begin{aligned}
\langle p_k, Ap_k\rangle &= \langle r_k + b_{k-1}p_{k-1}, Ap_k\rangle \\
&= \langle r_k, Ap_k\rangle \quad \text{by: } \mathcal{H}_2(k) \\
\langle r_k, p_k\rangle &= \langle r_k, r_k + b_{k-1}p_{k-1}\rangle \\
&= \langle r_k, r_k\rangle \quad \text{by: } \mathcal{H}_1(k)
\end{aligned} \tag{1.3}$$

The first is implied by $\mathcal{H}_2(k)$ and the second one is asserted by $\mathcal{H}_1(k)$. Next, we prove that $\mathcal{H}_3(k+1)$ is true.

$$\begin{aligned}
\langle r_{k+1}, r_k\rangle &= \langle r_k, r_k\rangle - a_k\langle r_k, Ap_k\rangle \qquad (1.4)\\
&= \langle r_k, r_k\rangle - a_k\langle p_k, Ap_k\rangle \quad \text{by (1.3)}\\
&= \langle r_k, r_k\rangle - \langle r_k, r_k\rangle\\
&= 0\\
\forall\, 0 \le j \le k-1: \quad \langle r_{k+1}, r_j\rangle &= \langle r_k - a_k A p_k, r_j\rangle\\
&= \langle r_k, r_j\rangle - a_k\langle Ap_k, r_j\rangle\\
&= -a_k\langle Ap_k, r_j\rangle\\
&= -a_k\langle Ap_k, p_j - b_{j-1}p_{j-1}\rangle\\
&= 0 \quad \text{by } \mathcal{H}_2(k)\\
&\implies \mathcal{H}_3(k+1) \text{ is true.}
\end{aligned}$$

Next, we consider:

$$\begin{aligned}
\langle r_{k+1}, p_k\rangle &= \langle r_k, p_k\rangle - a_k\langle Ap_k, p_k\rangle \qquad (1.5)\\
&= \langle r_k, r_k\rangle - a_k\langle Ap_k, p_k\rangle \quad \text{By: (1.3)}\\
&= 0\\
\forall\, 0 \le j \le k-1: \quad \langle r_{k+1}, p_j\rangle &= \langle r_k - a_k A p_k, p_j\rangle\\
&= \langle r_k, p_j\rangle - a_k\langle Ap_k, p_j\rangle\\
&= 0 \quad \text{by: } \mathcal{H}_1(k) \wedge \mathcal{H}_2(k)\\
&\implies \mathcal{H}_1(k+1) \text{ is true}
\end{aligned}$$

One last hyphothesis to prove. Consider:

$$\begin{aligned}
\langle p_{k+1}, Ap_k\rangle &= \langle r_{k+1}, Ap_k\rangle + b_k\langle p_k, Ap_k\rangle \qquad (1.6)\\
&= \langle r_{k+1}, Ap_k\rangle + \frac{\langle r_{k+1}, r_{k+1}\rangle}{\langle r_k, r_k\rangle}\langle p_k, Ap_k\rangle\\
&= \langle r_{k+1}, Ap_k\rangle + a_k^{-1}\langle r_{k+1}, r_{k+1}\rangle\\
&= \langle r_{k+1}, a_k^{-1}(r_k - r_{k+1})\rangle + a_k^{-1}\langle r_{k+1}, r_{k+1}\rangle\\
&= \langle r_{k+1}, -a_k^{-1}r_{k+1}\rangle + a_k^{-1}\langle r_{k+1}, r_{k+1}\rangle \quad \text{by } \mathcal{H}_3(k+1) \text{ from: (1.4)}\\
&= 0\\
\forall\, 0 \le j \le k-1: \quad \langle p_{k+1}, Ap_j\rangle &= \langle r_{k+1} + b_k p_k, Ap_j\rangle\\
&= \langle r_{k+1}, a_j^{-1}(r_j - r_{j+1})\rangle\\
&= 0 \quad \text{by: } \mathcal{H}_3(k+1)\\
&\implies \mathcal{H}_2(k+1) \text{ is true.}
\end{aligned}$$

All hypotheses fall through, and the base case is true by the algorithm. The proof has been completed.

Problem 2

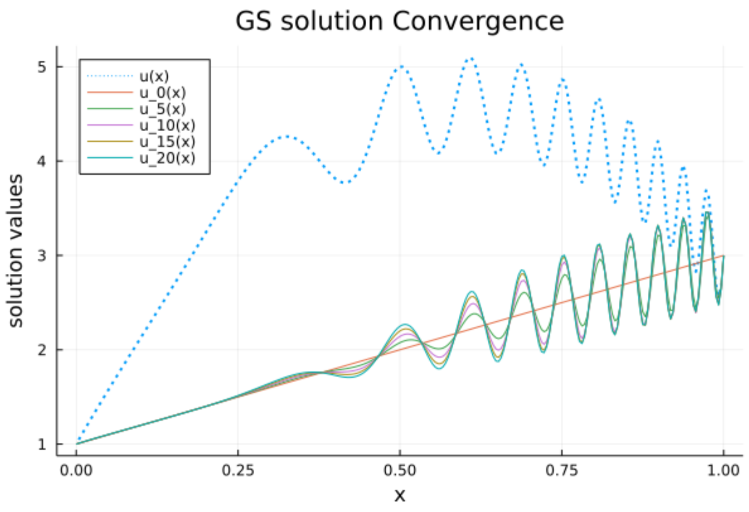
Problem 2 Code

```
"""
    Make a finite difference method for the following system:
        u''(x) = f(x) u(0) = a, u(1) = b.
    Dirichilet boundary conditions
    Parameter:
        n: grid point including the boundary point. The number of interval is
        n - 1, and the interval size is h = 1/(n - 1).
"""
function MakeFiniteDiffProblem(
    n;
    f::T1=nothing,
    a::T2=0,
    b::T2=0
) where {T1<:Union{Function, Nothing}, T2<:Number}
    @assert n >= 3 "The number of grid point including"*
        "the boundary conditions has to be at least 3"
    if isnothing(f)
        f = (x) -> 0
    end
    h = 1/(n - 1)
    gridPoints = LinRange(0, 1, n) |>  collect
    SymM = SymTridiagonal(fill(-2.0, n - 2), fill(1.0, n - 3))
    SymM ./= h^2
    RHS = f.(gridPoints[2:end - 1])
    RHS[1] -= a/h^2
    RHS[end] -= b/h^2
return SymM, RHS end


"""
    Unpreconditioned Conjugate Gradient Algorithm.
"""
function CGS(
    A,
    b;
    x0=nothing,
    eps=1e-10,
    maxitr=20,
    eps2=Inf
)
    if isnothing(x0)
        x0 = similar(b)
    end
    r = b - A*x0
    p = r
    v = Vector{typeof(x0)}()
    push!(v, x0)
    Ap = similar(b)
    for _ in 1:maxitr
        Ap .= A*p
        a = dot(r, r)/dot(p, Ap)
        if a < 0
            @warn ("Matrix is not symmetric positive definite for CGS. a=$(a)")
        end
        push!(v, v[end] + a*p)

        b = 1/dot(r, r)
        r = r - a*Ap

        if norm(r, Inf) < eps && norm(v[end] - v[end - 1], Inf) < eps2
            println("norm small as: $(norm(r, Inf))")
            break
        end
        b *= dot(r,r)
        p = r + b*p
    end

return hcat(v...) end


"""
    Performs the Guass Sediel Iterations.
"""
function GS(
    A,
    b;
    x0=nothing,
    eps=1e-10,
    maxitr=20,
    eps2=Inf
)
    U = triu(A, 1)
    L = tril(A)
    if isnothing(x0)
```

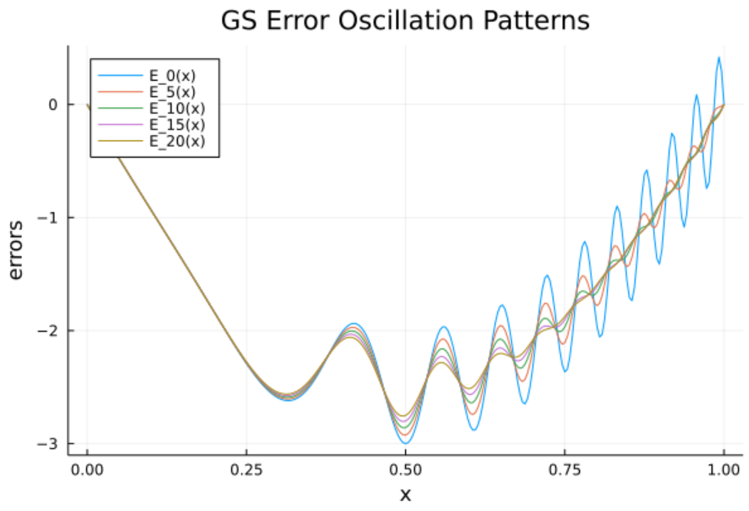
```
        x0 = zeros(size(b))
    end
    v = Vector{typeof(x0)}()
    push!(v, x0)
    for _ in 1:maxitr
        push!(v, L\(b - U*v[end]))
        res = norm(b - A*v[end], Inf)
        if norm(v[end] - v[end - 1], Inf) < eps2 &&
            res < eps
            break
        end
    end
return hcat(v...) end

### Using the code above to answer the questiosn we have -----------------------

using LinearAlgebra, Plots

function f(x)
    phi(x) = 20*pi*x^3
    Dphi(x) = 60*pi*x^2
    DDphi(x) = 129*pi*x
    a = 0.5
return -20 + a*DDphi(x)*cos(phi(x)) - a*(Dphi(x)^2)*sin(phi(x)) end

function f(x::AbstractVecOrMat)
return f.(x) end

function u(x)
    a = 0.5
    phi(x) = 20*pi*x^3
return 1 + 12*x - 10*x^2 + a*sin(phi(x)) end

function u(x::AbstractVecOrMat)
return u.(x) end


function Problem2GS(m=257)
    lbd = 1; rbd = 3
    A, b = MakeFiniteDiffProblem(m, f=f, a=lbd, b=rbd)
    # A |> display
    # b |> display
    GridPointsBd = (LinRange(0, 1, m) |> collect)
    IntGridPoints = GridPointsBd[2:end - 1]
    solns = GS(A, b, x0=1 .+ 2*IntGridPoints, maxitr=20)

    # Plotting the solutions along with the correct solutions.

    fig = plot(
        IntGridPoints,
        u(IntGridPoints),
        legend=:topleft,
        label="u(x)",
        line=(:dot, 2),
        xlabel="x",
        ylabel="solution values",
        title="GS solution Convergence"
    )
    for J in 1:5:size(solns, 2)
        plot!(
            fig, GridPointsBd, vcat(lbd, solns[:, J], rbd),
            label="u_$(J-1)(x)"
        )
    end
    fig |> display
    savefig(fig, "p2_gs_solns.png")

    fig = plot(
        legend=:topleft,
        label="u(x)",
        line=(:dot, 2),
        xlabel="x",
        ylabel="errors",
        title="GS Error Oscillation Patterns"
    )
    @info "Problem 2 GS Errors: "
    for J in 1:5:size(solns, 2)
        err = vcat(lbd, solns[:, J], rbd) - u(GridPointsBd)
        plot!(
            fig, GridPointsBd,
            err,
            label="E_$(J-1)(x)"
        )
        err = norm(err, Inf)/sqrt(m - 1)
        println("err: $(err)")
    end
    fig |> display
    savefig(fig, "p2_gs_errors.png")
end

function Problem2CG(m=257)
    lbd = 1; rbd = 3
    A, b = MakeFiniteDiffProblem(m, f=f, a=lbd, b=rbd)
    # A |> display
    # b |> display
    GridPointsBd = (LinRange(0, 1, m) |> collect)
    IntGridPoints = GridPointsBd[2:end - 1]
    solns = CGS(A, b, x0=1 .+ 2*IntGridPoints, maxitr=20)

    # Plotting the solutions along with the correct solutions.

    fig = plot(
        IntGridPoints,
        u(IntGridPoints),
        legend=:topleft,
        label="u(x)",
        line=(:dot, 2),
        xlabel="x",
        ylabel="solution values",
        title="CG solution Convergence"
    )
    for J in 1:5:size(solns, 2)
        plot!(
            fig, GridPointsBd, vcat(lbd, solns[:, J], rbd),
            label="u_$(J-1)(x)"
        )
    end
    fig |> display
    savefig(fig, "p2_cg_solns.png")

    fig = plot(
        legend=:topleft,
        label="u(x)",
        line=(:dot, 2),
        xlabel="x",
        ylabel="errors",
        title="CG Error Oscillation Patterns"
    )
    @info "Problem 2 CG Errors: "
    for J in 1:5:size(solns, 2)
        err = vcat(lbd, solns[:, J], rbd) - u(GridPointsBd)
        plot!(
            fig, GridPointsBd,
            err,
            label="E_$(J-1)(x)"
        )
        err = norm(err, Inf)/sqrt(m - 1)
        println("err: $(err)")
    end
    fig |> display
    savefig(fig, "p2_cg_errors.png")
return end

Problem2CG();
Problem2GS();
```

GS Solutions

Below is a plot of solutions generated by the Gauss Sediel method compare to the correct solution. 20 Iterations of GS is performed 0, 5, 10, 16, 20'th iterations are plotted:



And the plot of the errors for each of the solution is:

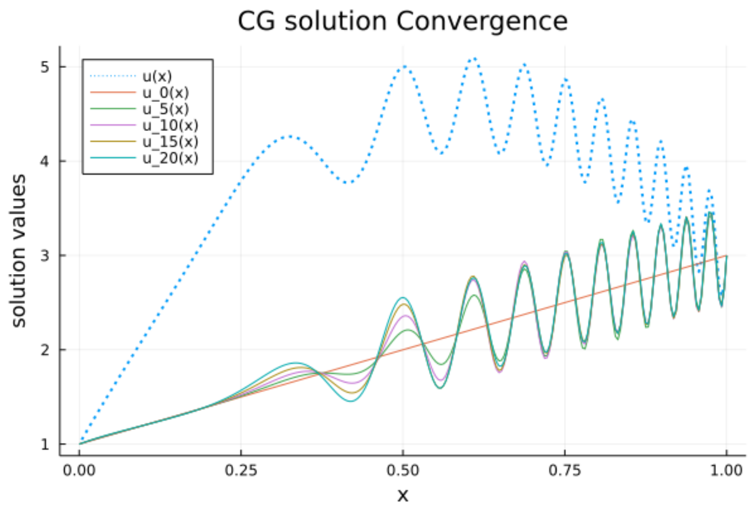


Please observe that the error is getting smoother and smoother as higher frequencies of oscilations are wiped away by the iterative methods. Finally, this is a list for the Inf Norm of the errors:

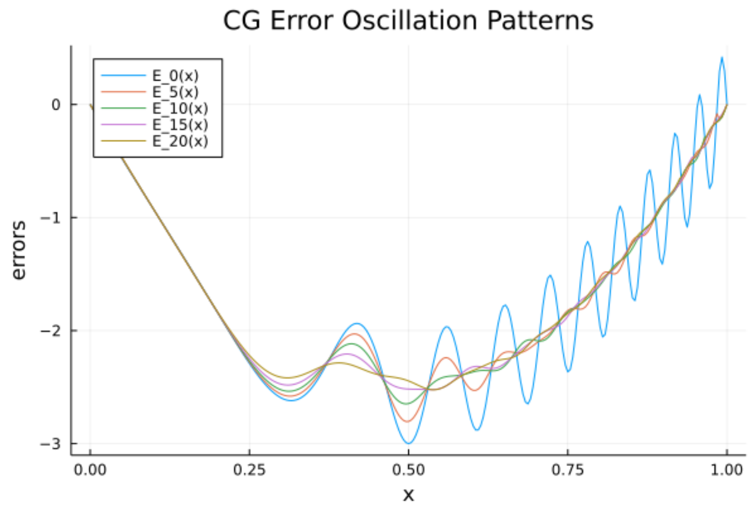
```
err: 0.1875
err: 0.1827841933736352
err: 0.1787560123586121
err: 0.1753054397389111
err: 0.17228048081525071
```

CG Solutions

The plot of the solution found by CG during iterations: 0, 5, 10, 15, 20 is:



The plot of the error of the solution found by CG for solution at the same iteration is plotted:



Observe that the same type of smoothing of the high frequencies oscilations of the solution exists for CG as well. Here is the list of errors for the solution found by cg at these key iterations:

```
err: 0.1875
err: 0.1753017032952856
err: 0.16554214673565795
err: 0.15765193211388995
err: 0.15774519163539247
```

Problem 3

Problem 3 Code

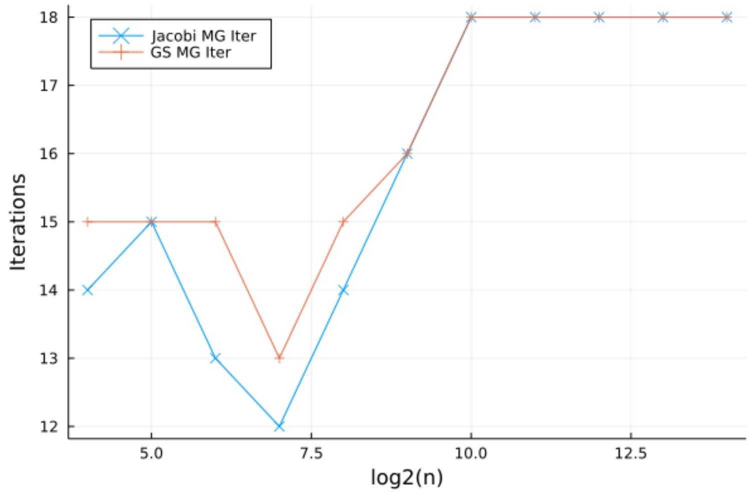
```
using LinearAlgebra


# ---- 1D Problem  Generation ---------------------------------------------------
"""
    Make a finite difference method for the following system:
        u’’(x) = f(x) u(0) = a, u(1) = b.
```

```julia
    Dirichilet boundary conditions
    Parameter:
        n: grid point including the boundary point. The number of interval is
        n - 1, and the interval size is h = 1/(n - 1).
"""
function MakeFiniteDiffProblem(
    n;
    f::T1=nothing,
    a::T2=0,
    b::T2=0
) where {T1<:Union{Function, Nothing}, T2<:Number}
    @assert n >= 3 "The number of grid point including"*
        "the boundary conditions has to be at least 3"
    if isnothing(f)
        f = (x) -> 0
    end
    h = 1/(n - 1)
    gridPoints = LinRange(0, 1, n) |>  collect
    SymM = SymTridiagonal(fill(-2.0, n - 2), fill(1.0, n - 3))
    SymM ./= h^2
    RHS = f.(gridPoints[2:end - 1])
    RHS[1] -= a/h^2
    RHS[end] -= b/h^2
return SymM, RHS end

function MakeFiniteDiffMatrix(n)
    A, _ = MakeFiniteDiffProblem(n)
return A end

# Iterative Solvers for linear system -----------------------------------------
function WeightedJacobiUpdate(
    A, b,
    x0;
    w=2/3
)
    D = reshape((diagm(A)./w).^(-1), length(x0), 1)
return D.*b + (I - D.*A)*x0 end


"""
    Performs the Guass Sediel Iterations.
"""
function GSUpdate(
    A,
    b,
    x0
)
    U = triu(A, 1)
    L = tril(A)
return L\(b - U*x0) end

function diagm(T::SymTridiagonal)
return T.dv end


# Multi-Grid related methods ---------------------------------------------------

function UpScaleWith(v)
    vFine = zeros(2*length(v) + 1)
    vFine[2:2:end - 1] = v
    vFine[1] = v[1]/2
    vFine[3:2:end - 2] = 0.5*(v[1:end - 1] + v[2: end])
    vFine[end] = v[end]/2
return vFine end

"""
    Give it an initial guess, it will modify it by performing
    one step of 2-Grid Multi-Grid method and returns the
    residual vector.
"""
function MultiGridOnce!(
    A,
    A_coarse,
    rhs,
    x0;
    solver::Function = WeightedJacobiUpdate,
    damp_itr=1
)
    for _ in 1:damp_itr
```

```
        x0 .= solver(A, rhs, x0)
    end
    r = rhs - A*x0

    # rCoarse = r[2:2:end - 1]
    rCoarse = view(r, 2:2:length(r) - 1)
    eCoarse = A_coarse\rCoarse
    eFine = UpScaleWith(eCoarse)
    x0 .+= eFine


return rhs - A*x0 end

function f(x)
    phi(x) = 20*pi*x^3
    Dphi(x) = 60*pi*x^2
    DDphi(x) = 129*pi*x
    a = 0.5
return -20 + a*DDphi(x)*cos(phi(x)) - a*(Dphi(x)^2)*sin(phi(x)) end


function f(x::AbstractVecOrMat)
return f.(x) end


function u(x)
    a = 0.5
    phi(x) = 20*pi*x^3
return 1 + 12*x - 10*x^2 + a*sin(phi(x)) end


function u(x::AbstractVecOrMat)
return u.(x) end


### Using the above function to solve problems ---------------------------------

using Plots

function PerformMultiGrid(n = 257, solver=WeightedJacobiUpdate)
    xgridWithBC = LinRange(0, 1, n) |> collect
    lbc = 0
    rbc = 0
    A, b = MakeFiniteDiffProblem(n, f=f, a=lbc, b=rbc)
    Ac = MakeFiniteDiffMatrix(div((n - 1), 2) + 1)
    x0 = zeros(size(b))
    TotalItr = 0
    for Itr in 1:500
        r = MultiGridOnce!(A, Ac, b, x0, solver=solver)
        relRes = norm(r)/norm(b)
        println(relRes)
        if relRes < 1e-4
            TotalItr = Itr
            println("total Itr: $(Itr)")
            break
        end
    end

return TotalItr end

function MultiGridConvergence()
    WJacobiItr = Vector{Int64}()
    GSItr =  Vector{Int64}()
    GridPointsCount = 2 .^ collect(4:14)
    for n in GridPointsCount
        n = n + 1
        push!(WJacobiItr, PerformMultiGrid(n, WeightedJacobiUpdate))
        push!(GSItr, PerformMultiGrid(n, GSUpdate))
    end
    fig = plot(
        GridPointsCount.|>log2, WJacobiItr,
        label="Jacobi MG Iter",
        marker=:x,
        xlabel="log2(n)", ylabel="Iterations",
        legend=:topleft
    )
    plot!(
        fig,
        GridPointsCount.|>log2, GSItr,
        label="GS MG Iter",
        marker=:+
    )
    fig |> display
    savefig(fig, "p3_mg_itr.png")

end

MultiGridConvergence();
```

The core part of the method is implemented in the “MultiGridOnce!” function.

The number of iterations underwent by the 2 Grid little v cycle is unrelated to the number of discretization points for the interval. Here is the plot to demonstrate this property of the algorithm.



The tolerance is measured in relative residual error and the tolerance is $1e-4$.

Problem 4

4(a)

With bounded norm, and the assumption that the initial error has 2-norm 1, and $\|I - M^{-1}A\|_1 = 1/2$, then it requires about 20 iterations to reduce the 2 norm of the error to less than 2^{-20}. However on computer it might not work because this is smaller than machine epsilon under IEEE Float64. Here is the derivation:

$$
\begin{aligned}
e_k &= (I - M^{-1}A)^k e_0 \qquad (5.a.1)\\
\|e_k\| &= \|(I - M^{-1}A)^k e_0\| \\
\|e_k\| &\le \|(I - M^{-1}A)^k\| \|e_0\| \\
\|e_k\| &\le \|I - M^{-1}A\|^k \|e_0\| \\
\frac{\|e_k\|}{\|e_0\|} &\le \|I - M^{-1}A\|^k \\
&\text{Substituting...} \\
\|e_k\|_2 &\le \frac{1}{2^k} < 2^{-20} \\
&\implies k > 20
\end{aligned}
$$

Done. However, if we were given that the spectral radius of the matrix $I - M^{-1}A$ is equal to 1/2, then there is no way to tell exactly how many iterations it would be needed to decrease the 2-Norm. One of the special case to consider is when the matrix $I - M^{-1}A$ is normal, and in that case the 2 norm coincided with the spectral radius of the matrix. On the other hand if the matrix is similar to a Jordan form, then raising the power of it will make the jordan block to be a upper triangular matrix consists of truncated Binomial expanions on each row. This makes the 2-Norm impossible to predict.

4(b)

The companion matrix together with some initial guess that makes the initial residual $r_0 = \mathbf{e}_n$ can halt the GMRes algorithm all the way until the final iteration.