3.1.1

using LinearAlgebra

using Plots

using Printf

#### NOTE 3.1.1

# T = (A^-1)\*B

A1 = [4 -1 0 -1 0 0 0 0 0;

-1 4 -1 0 -1 0 0 0 0;

0 -1 4 0 0 -1 0 0 0;

-1 0 0 4 -1 0 -1 0 0;

0 -1 0 -1 4 -1 0 -1 0;

0 0 -1 0 -1 4 0 0 -1;

0 0 0 -1 0 0 4 -1 0;

0 0 0 0 -1 0 -1 4 -1;

0 0 0 0 0 -1 0 -1 4]

B1 = [75; 0; 50; 75; 0; 50; 175; 100; 150]

T1 = A1\B1

Z1 = fill(NaN, (5,5))

for i = 1:3

for j = 1:3

Z1[i+1,j+1] = T1[3\*(i-1)+j]

end

end

Z1[:,1] = fill(75, (1,5))

Z1[:,5] = fill(50, (1,5))

Z1[5,:] = fill(100, (1,5))

Z1[1,:] = fill(0, (1,5))

contour(Z1,fill=true)

A close up of a logo

Description automatically generated

cond(A1)

λ = eigvals(A1)

k = maximum(λ)/minimum(λ)

### both condition numbers were 5.83

# x0 is initial guess: r0 = b - A\*x0

x0size = size(A1,2)

α = 1

ρ = α

ω = α

x0 = fill(5., (x0size,1)) |> vec

r0 = B1 - A1\*x0

r0hat = r0

check = dot(r0hat,r0)

ν = fill(0, (x0size,1)) |> vec

P = fill(0, (x0size,1)) |> vec

flag = 0

i = 1

num\_iter = 10

res = fill(NaN , (1,num\_iter-1)) |> vec

hfhdfhk = 1

while true

global (r0,r0hat,α,ρ,ω,x0,ν,p,flag)

global ρnew = dot(r0hat,r0)

global β = (ρnew/ρ)\*(α/ω)

global P = r0 + β\*(P-(ω\*ν))

global ν = A1\*P

global α = ρnew/dot(r0hat,ν)

global h = x0 + α\*P

global s = r0 - α\*ν

global t = A1\*s

global ω = dot(t,t)/dot(t,t)

global xnew = h + ω\*s

global r0 = s - ω\*t

global tol = norm(r0)

global x0 = xnew

global ρ = ρnew

global res[i] = tol

global i += 1

if tol < 1e-10

global flag = 1

break

end

if i >= num\_iter

global flag = -1

break

end

end

#### When wiki says "i" I am going to say "i+1"

#go to x0size+1

# also, r(i-1) is always going to be r0 in my code. I will update it at the end of the script

# If h is accurate enough, then set xi = h and quit

x = LinRange(1, 9, 9)

plot(x, res)

A picture containing text, sky

Description automatically generated

# NOTHING IS A MATRIX EXCEPT FOR A1

# also nothing needs to be kept two of except ρ

# compute norm of r

# r is the residual, so if the norm is small then we are good

#### NOTE 3.1.2

# n is going to change each iteration - for now, lets do n = 3

function InitializeArray(n)

prev\_middle = 500

outer\_counter = 1

iter = 0

tol = 1e-6

T = zeros(n+2, n+2)

# Apply the boundary conditions

T\_south = 0.

T\_east = 75.

T\_north = 100.

T\_west = 50.

T[:,1] .= T\_east

T[end,:] .= T\_north

T[:,end] .= T\_west

T[1,:] .= T\_south

# Note: I may want to fix the corners for plotting, but it's not needed for the solve.

# Loop until convergence

return T

end

T = InitializeArray(3)

contour(T,fill=true)

stop

# since variables inside a loop are local, then we need to use this 'let' stuff to pass in the stuff from before.

function GetContour(tol,iterMax,n)

T = InitializeArray(n)

flag = 0

ωopt = 2/(1+sin(π/n+1))

iter = 0

while flag == 0

# iter is 'global' since I want to reference it after the while loop ends. This is \*not\* needed if this while loop was inside a function.

iter += 1

for i = 2:n+1

for j = 2:n+1

T[i,j] = (1-ωopt)\*(T[i,j]) + ωopt\*(1/4\*(T[i-1,j] + T[i+1,j] + T[i,j-1] + T[i,j+1]))

end

end

# Compute R = Ax - b

resid = zeros(n+2,n+2)

for i = 2:n+1

for j = 2:n+1

resid[i,j] = -T[i,j] + 1/4\*(T[i-1,j] + T[i+1,j] + T[i,j-1] + T[i,j+1])

end

end

# Since resid is a matrix, I'll use a norm

if norm(resid) <= tol

flag = 1

break

elseif iter >= iterMax

flag = -1

error("Failed to converge")

break

end

end

return(T)

end

stop

T = GetContour(1e-5,1000,7)

contour(T,fill=true)

stop

function GetAccurateContour()

tol=1e-5

itermax=50000

Tmiddle=500

counter=0

MiddleTemp=fill(NaN,(1,100))

SizeOfMesh = fill(NaN,(1,100))

while true

counter += 1

n = counter\*2+1

T = GetContour(tol,itermax,n)

Tmiddlenew = T[counter+1,counter+1]

change = abs(Tmiddlenew - Tmiddle)

Tmiddle = Tmiddlenew

MiddleTemp[counter] = Tmiddle

SizeOfMesh[counter] = n

if counter > 75

return(T,MiddleTemp,counter,change,SizeOfMesh)

elseif change < tol

return(T,MiddleTemp,counter,change,SizeOfMesh)

end

end

end

T = GetAccurateContour()

contourf(T[1])

x = T[5] |> vec

y = T[2] |> vec

plot(x,y)

xlabel!("Length of Grid (Grid is square)")

ylabel!("Value of Middle Point")

A screenshot of a cell phone

Description automatically generated

A close up of a logo

Description automatically generated

3.2

using LinearAlgebra

function f(x,a)

# where x is a row of data points

# and a is a row vector of the coefficients

# such that y = a[1]x^n + a[2]x^n-1 + ... + a[n]

p = size(x,1)

n = size(a,1)

y = fill(0,(p,1))

for j = 1:p

for i = 1:n+1

y[j] += a[i]\*x[j]^(i-1)

end

end

return(y)

end

stop = 0

start = 0

function buildX( x, f, n)

# where x is a row of data points

# f is funtion handle of the funtion we want to curve fit

# n is the power of the last term we want the approximation of y to have

p = size(x,1)

X = fill(NaN,(p,n+1))

for j = 1:p

for i = 0:n

X[j,i+1] = x[j]^i

end

end

return(X)

end

stop = 0

n = 3

num\_data\_points = 5

x = LinRange(1,num\_data\_points,num\_data\_points) |> vec

check = buildX(x,f,n)

a = [2 -1]

check2 = f(x,a)

check3 = 2\*x

check4 =

for x = linspace()

### I did not finish this problem