This lab explores the use of finite difference methods to approximate the solution $y(x)$ to linear boundary value problems of the form

$$y'' = p(x)y' + q(x)y + r(x) \qquad \text{for} \qquad a \le x \le b$$

where $y(a) = \alpha$ and $y(b) = \beta$. Under the assumptions that $p$, $q$ and $r$ are continuous and moreover that $q > 0$, it is theoretically known there is a unique solution $y(x)$ to the differential equation. It therefore makes sense to try and approximate that solution numerically.

Begin by recalling that the derivative operator $D$ satisfies

$$D = \frac{1}{h}\log\mathcal{E} = \frac{2}{h}\log\left(\tfrac{1}{2}(\Delta_0 + \sqrt{\Delta_0^2 + 4I}\,)\right)$$

where $\mathcal{E}$ and $\Delta_0$ are respectively the shift and central difference operators

$$\mathcal{E}z_k = z_{k+1} \qquad \text{and} \qquad \Delta_0 z_k = z_{k+\frac{1}{2}} - z_{k-\frac{1}{2}}.$$

Since $\Delta_0 = \mathcal{O}(h)$ and assuming $h$ is sufficiently small, a power-series expansion of log yields

$$D^2 = \frac{4}{h^2}\sum_{j=0}^{\infty}\sum_{\ell=0}^{\infty}\frac{1}{2j+1}\binom{-1/2}{j}\frac{1}{2\ell+1}\binom{-1/2}{j}\left(\tfrac{1}{2}\Delta_0\right)^{2j+2\ell+2}.$$

Truncating the series to the leading term yields

$$D^2 = \frac{4}{h^2}\left(\tfrac{1}{2}\Delta_0\right)^2 = \frac{1}{h^2}\Delta_0^2 + \mathcal{O}(h^2).$$

This approximation will represent $y''$.

For the first derivative recall the formula

$$D = \frac{1}{h}\Delta_0\Upsilon_0\sum_{j=0}^{\infty}\sum_{\ell=0}^{\infty}\binom{-1/2}{j}\frac{1}{2\ell+1}\binom{-1/2}{\ell}\left(\tfrac{1}{2}\Delta_0\right)^{2j+2\ell}$$

where $\Upsilon_0$ is the averaging operator

$$\Upsilon_0 z_k = \tfrac{1}{2}\left(z_{k+\frac{1}{2}} + z_{k-\frac{1}{2}}\right).$$

Again truncate the series to one term to obtain

$$D = \frac{1}{h}\Delta_0\Upsilon_0 + \mathcal{O}(h^2).$$

This shall be used to approximate $y'$.

**Discretizing the Differential Equation**

To discretize the differential equation divide the domain $[a,b]$ into $m+1$ equal pieces of size $h = (b-a)/(m+1)$. Consider the grid points $x_k = a+hk$ for $k = 1,\ldots,m$. Let $y_k$ be approximation of the exact solution $y(x_k)$ at each grid point. Note the boundary conditions imply $y_0 = \alpha$ and $y_{m+1} = \beta$.

Having defined $y_k$, now employ the approximations

$$y''(x_k) \approx \frac{1}{h^2}\Delta_0^2 y_k \qquad \text{and} \qquad y'(x_k) \approx \frac{1}{h}\Delta_0\Upsilon_0 y_k$$

to write the differential equation as the difference equation

$$\frac{1}{h^2}\Delta_0^2 y_k = p(x_k)\frac{1}{h}\Delta_0\Upsilon_0 y_k + q(x_k)y_k + r(x_k)$$

for $k = 1,2,\ldots,m$. Since

$$\frac{1}{h^2}\Delta_0^2 y_k = \frac{y_{k+1} - 2y_k + y_{k-1}}{h^2} \quad \text{and} \quad \frac{1}{h}\Delta_0\Upsilon_0 y_k = \frac{y_{k+1} - y_{k-1}}{2h},$$

then writing $p_k = p(x_k)$, $q_k = q(x_k)$ and $r_k = r(x_k)$ yields

$$\frac{y_{k+1} - 2y_k + y_{k-1}}{h^2} = p_k\frac{y_{k+1} - y_{k-1}}{2h} + q_k y_k + r_k.$$

Equivalently,

$$-y_{k+1} + 2y_k - y_{k-1} + \tfrac{h}{2}p_k(y_{k+1} - y_{k-1}) + h^2 q_k y_k = -h^2 r_k$$

for $k = 1,2,\ldots,m$.

The equations when $k = 1$ and $k = m$ can be rewritten involving the boundary conditions $y_0 = \alpha$ and $y_{m+1} = \beta$ as

$$-y_2 + 2y_1 + \tfrac{h}{2}p_1 y_2 + h^2 q_1 y_1 = \alpha + \tfrac{h}{2}p_1\alpha - h^2 r_1.$$

and

$$2y_m - y_{m-1} - \tfrac{h}{2}p_m y_{m-1} + h^2 q_m y_m = \beta - \tfrac{h}{2}p_m\beta - h^2 r_m.$$

The terms with $y_k$ have been written on the left and the boundary terms placed on the right. The result is a system of $m$ linear equations in the $m$ unknowns given by $y_k$ for $k = 1, 2, \ldots, m$.

To solve for the $y_k$, write the system in matrix form as $Ay = c$ where

$$A = \begin{bmatrix} 2+h^2q_1 & -1+\frac{h}{2}p_1 & 0 & \cdots & 0 \\ -1-\frac{h}{2}p_2 & 2+h^2q_2 & -1+\frac{h}{2}p_2 & 0 & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & 0 & -1-\frac{h}{2}p_{m-1} & 2+h^2q_{m-1} & -1+\frac{h}{2}p_{m-1} \\ 0 & \cdots & 0 & -1-\frac{h}{2}p_m & 2+h^2q_m \end{bmatrix}$$

and

$$c = (\alpha + \tfrac{h}{2}p_1\alpha - h^2r_1, -h^2r_2, \ldots, -h^2r_{m-1}, \beta - \tfrac{h}{2}p_m\beta - h^2r_m).$$

**Sparse Matrices**

The matrix $A$ corresponding to the finite difference scheme has a lot of zeros. In particular only the diagonal, subdiagonal and supradiagonal entries contain coefficients of the system. For an $m \times m$ matrix this means that $3m - 2$ entries out of $m^2$ are non-zero. For example, if $m = 128$ then

$$\frac{3m-2}{m^2} = \frac{382}{16384} \approx 2.3\,\text{percent}$$

of the entries are non-zero. This is called a sparse matrix.

As it would be wasteful to store all those zeros in memory and even more wasteful to compute with them, Julia includes a library for working such matrices called `SparseArrays`.

One way create a sparse matrix in Julia is to first create a regular matrix `A` and then convert it to a sparse matrix by removing the zeros with `A=sparse(A)`. This technique is demonstrated as follows:

```
julia> using LinearAlgebra,SparseArrays
```

```
julia> A=diagm([1.0,2,3,4])
4×4 Matrix{Float64}:
 1.0  0.0  0.0  0.0
 0.0  2.0  0.0  0.0
 0.0  0.0  3.0  0.0
 0.0  0.0  0.0  4.0

julia> A=sparse(A)
4×4 SparseMatrixCSC{Float64, Int64} with 4 stored entries:
 1.0   ⋅    ⋅    ⋅
  ⋅   2.0   ⋅    ⋅
  ⋅    ⋅   3.0   ⋅
  ⋅    ⋅    ⋅   4.0
```

There is an obvious drawback to this technique because the initial step of constructing the matrix could take too much memory. A better way constructs the sparse matrix directly by specifying only the non-zero entries. This can by done with `sparse(xs,ys,axy)` which creates a matrix with entries $a_{ij}$ such that

$$a_{ij} = \begin{cases} \texttt{axy[k]} & \text{for } i = \texttt{xs[k]} \text{ and } j = \texttt{ys[k]} \\ 0 & \text{otherwise.} \end{cases}$$

For example,

```
julia> A=sparse(1:4,1:4,[1.0,2,3,4])
4×4 SparseMatrixCSC{Float64, Int64} with 4 stored entries:
 1.0   ⋅    ⋅    ⋅
  ⋅   2.0   ⋅    ⋅
  ⋅    ⋅   3.0   ⋅
  ⋅    ⋅    ⋅   4.0
```

constructs the same sparse matrix but without the intermediate step that requires allocating the memory needed for a full $m \times m$ array.

Sometimes, specifying lists of indices and values is tricky. A compromise solution is to first declare a sparse matrix with no non-zero entries and then fill in the needed values using an assignment within a loop.

```
julia> A=spzeros(4,4)
4×4 SparseMatrixCSC{Float64, Int64} with 0 stored entries:
  ⋅    ⋅    ⋅    ⋅ 
  ⋅    ⋅    ⋅    ⋅ 
  ⋅    ⋅    ⋅    ⋅ 
  ⋅    ⋅    ⋅    ⋅ 

julia> for i=1:4
           A[i,i]=i
       end

julia> A
4×4 SparseMatrixCSC{Float64, Int64} with 4 stored entries:
 1.0   ⋅    ⋅    ⋅ 
  ⋅   2.0   ⋅    ⋅ 
  ⋅    ⋅   3.0   ⋅ 
  ⋅    ⋅    ⋅   4.0
```

## Approximating a Solution

We are now ready to approximate a solution to a linear two-point boundary-value problem using finite differences.

Each person will solve a different boundary value problem. In particular, your individualized problem will consist of different values $a$, $b$, $\alpha$, and $\beta$ for the boundary conditions and different functions $p(x)$, $r(x)$ and $q(x)$ for the differential equation. Click on the following link to retrieve your boundary value problem

    https://fractal.math.unr.edu/~ejolson/467-23/ab/mkab.cgi

Please do not use anyone else's differential equation for this lab.

The rest of this lab consists of a walk through demonstrating how to use the finite differences to approximate a solution of this differential equation. When I clicked on the link I obtained

Your differential equation is

$$y'' = p(x)\,y' + q(x)\,y + r(x) \qquad \text{where} \qquad a \le x \le b$$

with

```
p(x)=-0.80
q(x)=1.18+0.27*x^2
r(x)=sin(1.43*x)
a=0.92
b=3.05
```

and boundary conditions

```
y(a)=0.82
y(b)=0.22
```

Create a subdirectory called `lab05` and create the file `finite.jl` in that directory using `gedit` or some other program editor. Begin by defining the functions and parameters in the problem. Also load the `LinearAlgebra` and `SparseArrays` libraries as we will be using them later.

```julia
using LinearAlgebra,SparseArrays

p(x)=-0.80
q(x)=1.18+0.27*x^2
r(x)=sin(1.43*x)
a=0.92
b=3.05
alpha=0.82
beta=0.22
```

When performing numerical computations, it can be easy to lose track of what output corresponds to which input. Let's use the `Symbolics` library to print out the details of the differential equation before solving it.

```julia
using Symbolics

```

```
@variables x
println("p(x)=",p(x))
println("q(x)=",q(x))
println("r(x)=",r(x))
println("y($a)=",alpha)
println("y($b)=",beta)
```

At this point it would be reasonable to test the program by opening a terminal window, changing to the `lab05` subdirectory, starting Julia and then typing `include("finite.jl")`. The output should look similar to

```
julia> include("finite.jl")
p(x)=-0.8
q(x)=1.18 + 0.27(x^2)
r(x)=sin(1.43x)
y(0.92)=0.82
y(3.05)=0.22
```

Again, including the problem being solved as part of the output helps avoid errors. While there is not much room for getting the output of one program confused with another in a laboratory activity such as this one, such things are surprisingly important in practice.

Next, specify how many grid points will be used for the computation. Take $m = 32$ which is hopefully large enough. For extra credit you may perform a convergence study for your problem by repeating the calculation for different values of $m$ and checking how fast the solution converges as $m$ increases. What is the observed order of convergence?

We now create the matrix $A$ and vector $c$ needed for the equation $Ay = c$ to solve for $y$. Since $c$ is easier to construct first do that.

```
m=32
h=(b-a)/(m+1)
x=a.+(1:m)*h
c=-h^2*r.(x)
c[1]+=alpha+h/2*p(x[1])*alpha
c[m]+=beta-h/2*p(x[m])*beta
```

Note that `c` is built in stages. Line 24 initializes the entire vector with the term $-h^2 r_k$. Then line 25 adds the boundary term $\alpha + \frac{h}{2}p_1\alpha$ to the first entry and line 26 adds $\beta - \frac{h}{2}p_m\beta$ to the last.

The matrix $A$ may be created in a similar way by initializing the diagonal and then adding supradiagonal and subdiagonal with loops.

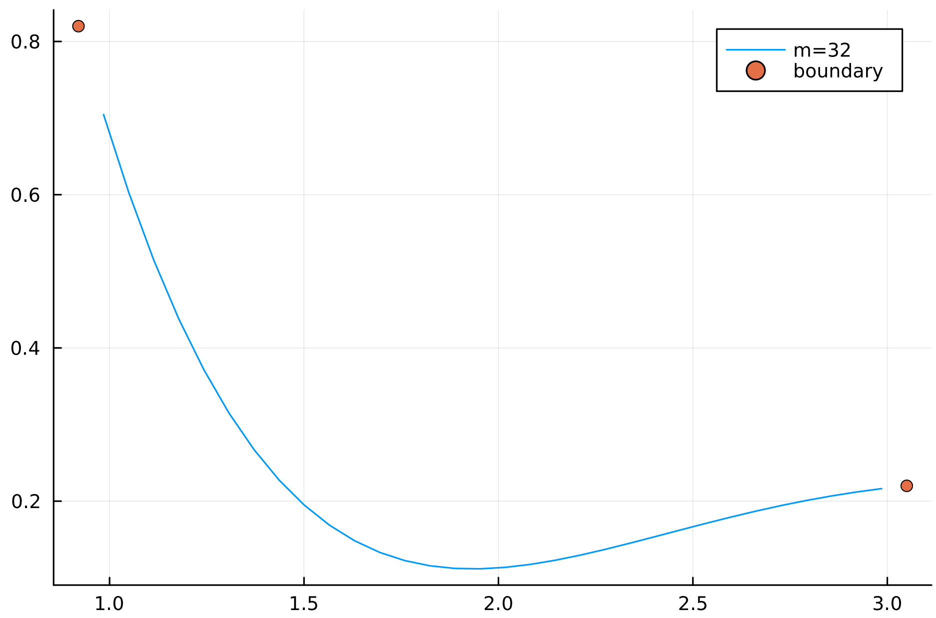
```
A=sparse(1:m,1:m,2.0.+h^2*q.(x))
for i=1:m-1
    A[i,i+1]=-1+h/2*p(x[i])
end
for i=2:m
    A[i,i-1]=-1-h/2*p(x[i])
end
```

Finding the approximation $y_k$ for $k = 1, 2, \ldots, m$ can now be performed with the command `y=A\c` which uses the built-in matrix libraries of Julia to efficiently solve the sparse linear algebra problem.

To finish this lab please print out the value of $y_{16}$ and draw a graph of the final approximation for your individualized differential equation. Code to do this for the example problem follows:

```
y=A\c
println("y(",x[16],")=",y[16])
using Plots
plot(x,y,label="m=$m")
scatter!([a,b],[alpha,beta],label="boundary")
```

Once everything works, use the command `savefig("graph05.pdf")` to save the graph. It should look similar to




If your graph looks *exactly* like the above figure, that may mean you forgot to change the differential equation to the individual one downloaded from the link mentioned earlier.

### Submitting Your Work

Two things should be uploaded for grading:

- A PDF file `lorenz.pdf` containing the code `finite.jl` and the output from running that code.
- The graph `graph05.pdf` corresponding to your boundary value problem.

The file `graph05.pdf` can be created by adding `savefig("graph05.pdf")` to the end of your program. The only thing left is to convert `lorenz.jl` and its output into a PDF file for upload. In the lab the commands

```
$ julia finite.jl >finite.out
$ j2pdf -o finite.pdf finite.jl finite.out
```

may be used to produce a file `finite.pdf` suitable for uploading. You may check your submission using `evince` to view the PDF files.

Before leaving don't forget to close the applications open on your desktop and logout. Exit the Julia REPL by typing ⟨ctrl⟩-`d` and then ⟨ctrl⟩-`d` again to close the terminal. The editor has a menu at the top. If using one of the lab computers, please reboot it into Microsoft Windows.