Turning a boundary-Value Problem into a Matrix Problem

Let's imagine we are presented with a boundary-value problem involving a second-order linear differential equation (of the sort you came to know and love in MATH 210):

$$y'' + f(x)y' + g(x)y = 0, \quad a \le x \le b, \tag{1}$$

with $y(a) = y_a$ and $y(b) = y_b$. We can recast this as a matrix equation as follows. First we discretize the problem by defining an array of size $N + 1$ in x, with $x_0 = a, x_N = b$, with uniform spacing $h = (b - a)/N$. In `numpy` parlance, we would say

```
x = numpy.linspace(a, b, N)
```

Denoting $y(x_n)$ by y_n, we can write centered (2nd-order accurate) expressions for the derivatives:

$$\begin{aligned} y'' &\approx \frac{y_{n+1} - 2y_n + y_{n-1}}{h^2} \\ y' &\approx \frac{y_{n+1} - y_{n-1}}{2h}, \end{aligned}$$

so, substituted and multiplying across by h^2, we have

$$y_{n+1} - 2y_n + y_{n-1} + \tfrac{1}{2}hf_n\left(y_{n+1} - y_{n-1}\right) + h^2 g_n y_n = 0,$$

where $f_n = f(x_n), g_n = g(x_n)$. Collecting terms, we find

$$\left(1 + \tfrac{1}{2}hf_n\right) y_{n+1} + \left(-2 + h^2 g_n\right) y_n + \left(1 - \tfrac{1}{2}hf_n\right) y_{n-1} = 0$$

for $n = 1, \ldots, N - 1$. We can't apply this for $n = 0$ or $n = N$ because y_{-1} and y_{N+1} are undefined, but at the ends of the range we can use the boundary conditions, $y_0 = y_a, y_N = y_b$.

With $\alpha_n = 1 + \frac{1}{2}hf_n, \beta_n = -2 + h^2 g_n, \gamma_n = 1 - \frac{1}{2}hf_n$, for $n = 1, \ldots, N - 1$ we have

$$\begin{array}{rcrcrcrcrcl} \beta_1\, y_1 & + & \alpha_1\, y_2 & & & & & & & = & -\gamma_1\, y_a \\ \gamma_2\, y_1 & + & \beta_2\, y_2 & + & \alpha_2\, y_3 & & & & & = & 0 \\ & & \gamma_3\, y_2 & + & \beta_3\, y_3 & + & \alpha_3\, y_4 & & & = & 0 \\ \cdot & & & & & & & & & & \cdot \\ \cdot & & & & & & & & & & \cdot \\ \cdot & & & & & & & & & & \cdot \\ & & & & & & & \gamma_{N-1}\, y_{N-2} & + & \beta_{N-1}\, y_{N-1} & = & -\alpha_{N-1}\, y_b \end{array}$$

This is a matrix equation

$$\mathbf{A}\mathbf{y} = \mathbf{r}$$

where $\mathbf{y}$ is the $N - 1$ dimensional column vector $(y_1, y_2, \ldots, y_{N-1})^T$, $\mathbf{r} = (-\gamma_1 y_a, 0, \ldots, 0, -\alpha_{N-1} y_b)^T$, and the matrix $\mathbf{A}$ is

$$\mathbf{A} = \begin{pmatrix} \beta_1 & \alpha_1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \gamma_2 & \beta_2 & \alpha_2 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \gamma_3 & \beta_3 & \alpha_3 & 0 & \cdots & 0 & 0 & 0 \\ \cdot & & & & & & & & \\ \cdot & & & & & & & & \\ \cdot & & & & & & & & \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & \gamma_{N-1} & \beta_{N-1} \end{pmatrix}$$

This equation is easily solved using the `scipy.linalg` function `solve`:

```
y = solve(A, r)
```

Note that there is nothing special about homogeneous equations. The procedure for an inhomogeneous system, with a function $s(x)$ on the right-hand side of Equation (1), is almost identical and can be solved the same way, except that now $\mathbf{r} = (h^2 s(x_1) - \gamma_1 y_a, h^2 s(x_2), \ldots, h^2 s(x_{N-2}), h^2 s(x_{N-1}) - \alpha_{N-1} y_b)^T$.