Technical University of Denmark

02685 Scientific Computing for Differential Equations
2017

Assignment 2

Authors:
Miguel Suau de Castro (s161333)
Michal Baumgartner (s161636)
March 16, 2017

Contents

1 2 point Boundary Value Problems

2-point Boundary Value Problems (BVP) represent a special case of differential equations where the solution is limited by a pair of constraints at both ends of the interval. Hence, the task consists of finding the solution that apart from solving the differential equation satisfies the boundary conditions.

In the linear case, a numerical approximation to the solution to these problems can be obtained solving a linear system. However, for nonlinear differential equations the solution is not straightforward and an iterative process is needed. In this exercise, we shall study different methods for solving nonlinear BVPs by applying them to the following differential equation:

$$\begin{aligned} \epsilon u(t)'' + u(t)(u(t)' - 1) = 0 \qquad & 0 \leq t \leq 1 \\ u(0) = \alpha \qquad & u'(1) = \beta \end{aligned} \tag{1}$$

1.1 Newton's method

If the independent variable t is discretized and the second derivative in equation 1 is replaced by the centered difference method, an approximation to the solution at a series of equidistant points can be found solving the following scheme:

$$\begin{aligned} \epsilon \left(\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} \right) + u_i \left(\frac{u_{i+1} - u_{i-1}}{2h} - 1 \right) = 0 \\ u(0) = \alpha \qquad u(1) = \beta \end{aligned} \tag{2}$$

Where $i = 1, 2, ..., m$ and h is the distance between points in the interval.

The centered approximation comes from a Taylor series expansion truncated to the second term. We can show that the approximation is second-order accurate by computing the local error:

$$\begin{aligned} \tau_i = \epsilon \left(\frac{u(t_{i-1}) - 2u(t_i) + u(t_{i+1})}{h^2} \right) + u_i \left(\frac{u(t_{i+1}) - u(t_{i-1})}{2h} - 1 \right) - \\ u(t_i)'' - u(t_i)(u(t_i)' - 1) \end{aligned} \tag{3}$$

The above expression is just the difference between the given differential equation and its finite difference approximation. By applying the Taylor series expansion to the first two terms:

$$\begin{aligned} \tau_i = \epsilon \left(u''(t_i) + \frac{1}{12} h^2 u'''(t_i) + O(h^4) \right) + u(t_i) \left(u'(t_i) + \frac{1}{6} h^2 u'''(t_i) + O(h^5) - 1 \right) - \\ u(t_i)'' - u(t_i)(u(t_i)' - 1) \end{aligned} \tag{4}$$

$$\tau_i = \frac{1}{12}h^2u'''(t_i) + O(h^4) + u(t_i)(\frac{1}{6}h^2u'''(t_i) + O(h^5)) \tag{5}$$

The dominant term in equation 5 is then dependent of h^2 and thus we can conclude that the local truncation error is order $O(h^2)$.

Equation 6 can be expressed as a nonlinear system of the form:

$$G(U) = 0 \tag{6}$$

And its roots can be obtained using Newton's method.

$$U^{[k+1]} = U^{[k]} + J(U^{[k]})^{-1}G(U^{[k]}) \tag{7}$$

Where $U^{[k]}$ is a vector containing a discrete approximation to the solution after k iterations and $J(U^{[k]})$ is the Jacobian matrix of the system, which in our case:

$$J(U) = \begin{cases} \frac{\epsilon}{h^2} - \frac{u_i}{2h} & j = i-1 \\ -\frac{2\epsilon}{h^2} - \frac{u_{i+1}-u_{i-1}}{2h} - 1 & j = i \\ \frac{\epsilon}{h^2} + \frac{u_i}{2h} & j = i-1 \end{cases} \tag{8}$$

Our implementation of the Newton's method consist of two functions: FunJac, which builds the vector G and the tridiagonal matrix J and NewtonsMethod, which updates the solution applying 7 until the tolerance is satisfied or the number of iterations is greater than the threshold.

The convergence and accuracy of the method depend on the proximity of the initial guess to the real solution. An approximation to the solution is given by equation 2.105 in LeVeque. Since an arbitrary choice might lead to non-convergence we have used this approximation as the initial value. It is shown in section 2.16.3 in Randall Leveque that the method is stable if the inverse of the Jacobian matrix is bounded. Figure 1 shows the initial guess along with the solution returned by the Newton's method for 50 grid points.

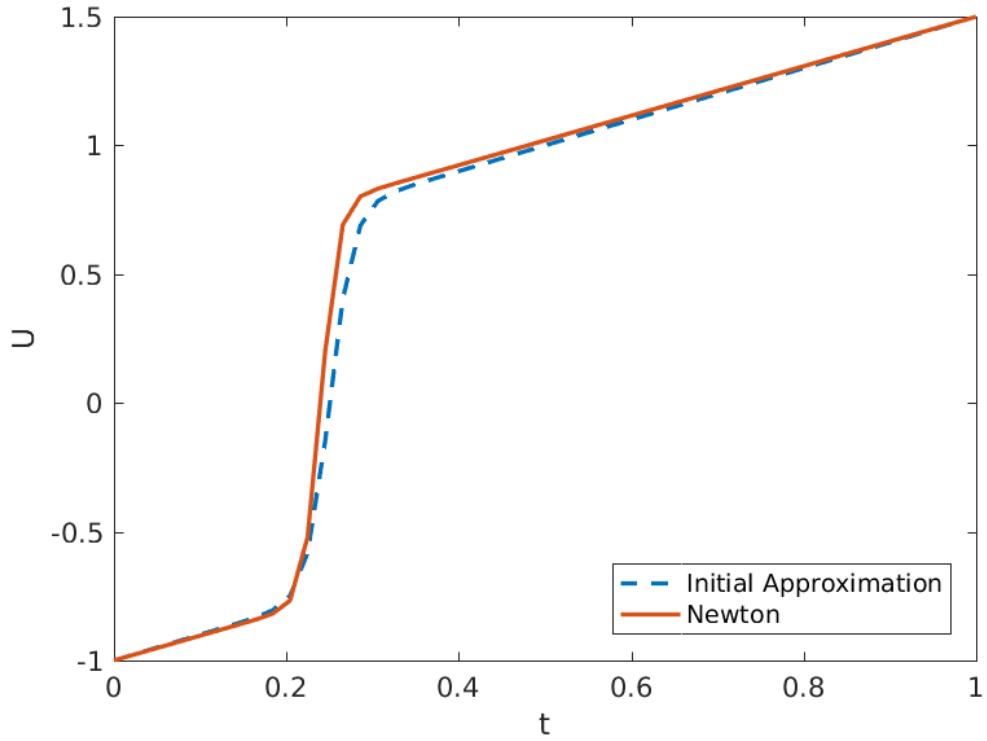

Figure 1: Initial approximation and solution to the BVP using Newton's method for 50 grid points

The solution to 1 has also been computed using bvp4c function in MATLAB. Figure 2 shows a comparative of the results obtained using Newton's method and bvp4c setting the tolerance equal to 0.01 and 33 grid points.

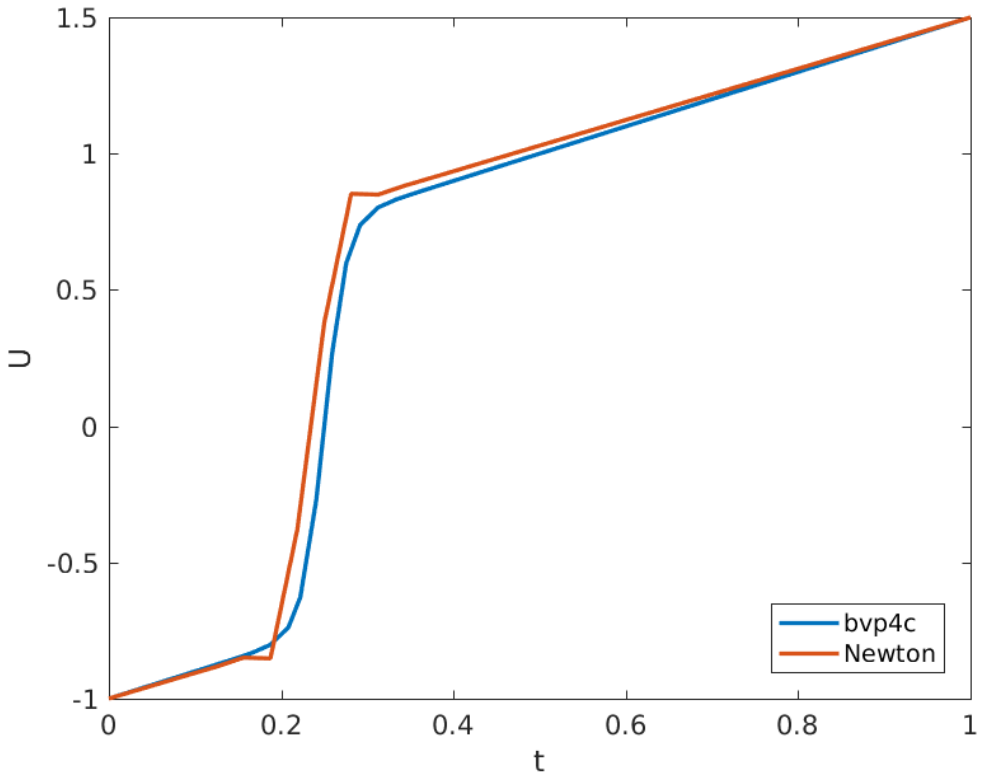

Figure 2: Initial approximation and solution to the BVP using Newton's method and bvp4c with 33 grid points

Finally a more accurate approximation has been obtained setting the tolerance of bvp4c to 10^{-10}. A section of the solution is shown in figure 3 along with the Newton's method solution now for 100 grid points. The region shown in the graph corresponds to a layer where the solution changes rapidly. It is easy to see that Newton's method lies further from the true solution in this region. The number of points required to obtain a more accurate solution is greater than anywhere else due to this sudden variation. Hence, better results could be obtained using a non-uniform grid. The solution would be more finely discretized in the mentioned region and thus any rapid change could be easily controlled.

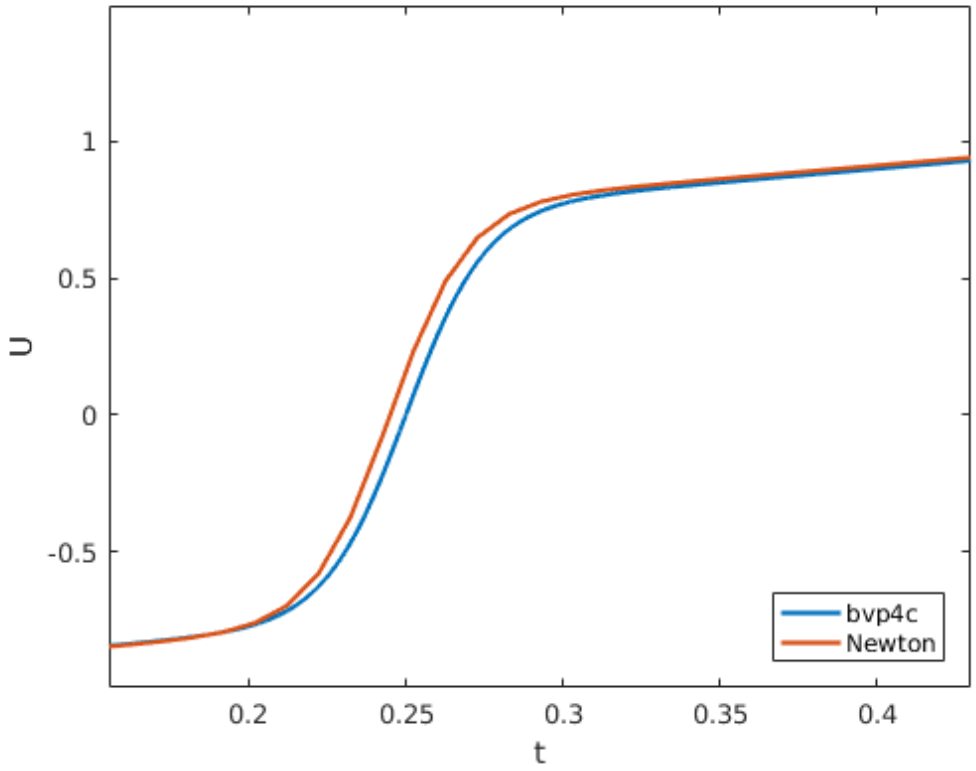

Figure 3: Section of the solution to the BVP using Newton's method with 100 grid points and bvp4c for a tolerance of 10^{-10}

1.2 Single shooting

One could also write the BVP in equation 1 as an initial value problem (IVP) of the form:

$$\epsilon\left(\frac{u_{i-1}-2u_i+u_{i+1}}{h^2}\right)+u_i\left(\frac{u_{i+1}-u_{i-1}}{2h}-1\right) \qquad (9)$$
$$u(0)=\alpha \qquad u'(0)=\sigma$$

Note that a new variable σ has been included whose value must be chosen so that the second constraint in 1 is satisfied:

$$r(\sigma)=U(1,\sigma)-\beta=0 \qquad (10)$$

An approximation to the solution of the IVP can be computed in MATLAB using the function ode45. The naive way to tackle the problem would be to try

different values of σ and choose the one that yields the closest solution to the boundary constraint. However, the use of a numerical algorithm to find the root of 10 would be computationally more convenient.

As a first approach we implemented the bisection method. Given two values of σ for which equation 10 has opposite sign we know that the solution will lie within the interval described by these two points. Consequently the solution is found by progressively reducing the interval until its size is below a tolerance. Although the algorithm will converge no matter the size of the interval, the problem now is translated into obtaining the two initial points. Moreover, the method presents a poor linear rate of convergence.

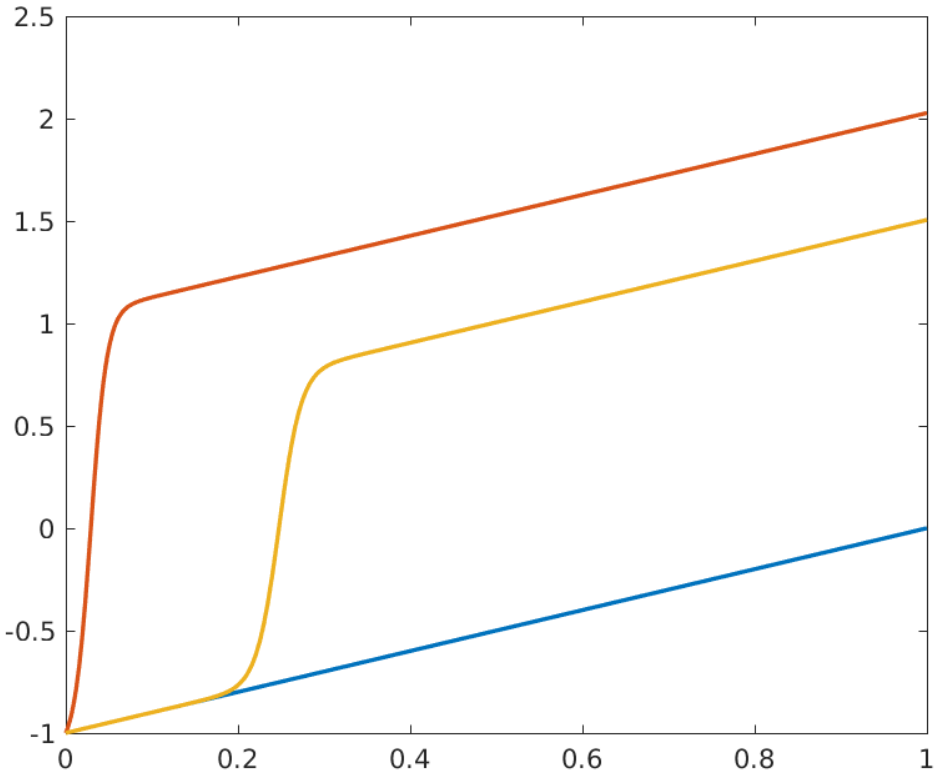

Figure 4: Solution to the IVP using the bisection method. U_a and U_b are the two initial guess of opposite sign while U_c is the solution that verifies the second boundary constraint

Figure 4 shows the solution obtained U_c applying the bisection method along with the two initial guesses U_a and U_b, of opposite sign with respect to the bound-

ary constraint. We see that after a certain number of iterations the algorithm is able to hit the second constraint.

A more efficient algorithm can be obtained applying Newton's method. For that, we would like to have an expression for the first partial derivative of 10 with respect to σ so that the value of σ can be updated in every iteration.

Since β in 10 is a constant we can write:

$$\frac{\partial r(\sigma)}{\partial \sigma} = \frac{\partial u(1,\sigma)}{\partial \sigma} \tag{11}$$

taking the partial derivative with respect to σ in 9:

$$\begin{gathered} \frac{\partial u''}{\partial \sigma} = \frac{-u'+1}{\epsilon}\frac{\partial u}{\partial \sigma} - \frac{u}{\sigma}\frac{\partial u'}{\partial \sigma} \\ \frac{\partial u(0)}{\partial \sigma} = 0 \qquad \frac{\partial u'(0)}{\partial \sigma} = 1 \end{gathered} \tag{12}$$

and since this is also an IVP it can be solved together with 9 as a single differential equation. As we mentioned before, Newton's method requires a suitable initial guess to converge. We experienced in this case that the algorithm is unable to converge for small values of ϵ in such cases, the bisection method should be applied. Figure 5 shows the solution at the end of the interval for a series of σ values. Although a more detailed reason for the non-convergence of the method is given in the next section, it easy to see that the function is close to be non-differentiable in a neighborhood of σ optima.

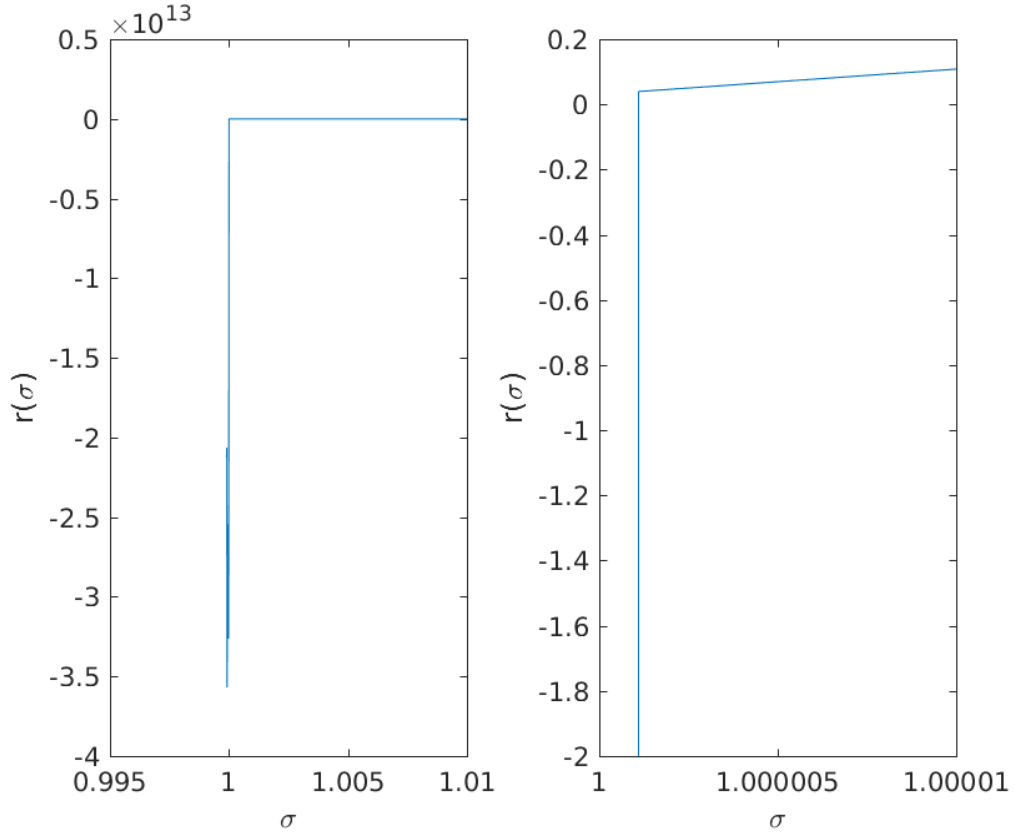

Figure 5: Solution to the IVP at the last point of the interval for a series of σ values close to the optima with $\epsilon = 0.01$

The rate of convergence of both algorithms has been plotted and can be seen in figure 6. The graph shows that for ϵ sufficiently large Newton's method converges quadratically.

Method	Iterations	CPU-time
Bisection	8	0.0054
Newton	3	0.0022
Secant	3	0.0036

Table 1: Number of iterations and CPU-time of all three algorithms with $\epsilon = 0.8$

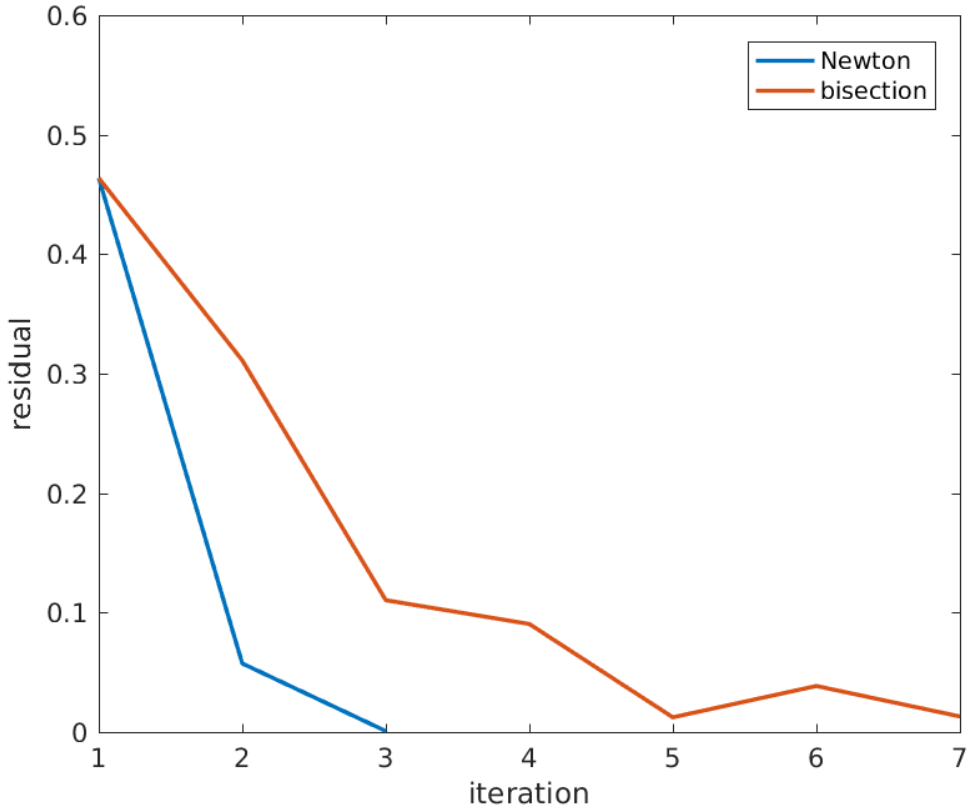

Figure 6: Newton and bisection algorithm rate of convergence for $\epsilon = 0.8$

Finally the secant method is a variation of the Newton's algorithm where the derivative is approximated by finite difference so that there is no need for solving an extra differential equation. The code for all three algorithms is collected in the appendix. Besides, some information about the three methods performance is shown in table 1.

ϵ	S_σ
0.01	1.0299×10^5
0.05	0.5791
0.1	0.3620
0.5	0.7110

Table 2: Number of iterations and CPU-time of all three algorithms with $\epsilon = 0.8$

1.3 Sensitivity analysis and non-convergence

In order to find an approximation to the partial derivative of the solution with respect to σ we can either solve equation **??** or apply the finite difference method. In both cases we see that for small values of ϵ the solution becomes very sensible to small variations of σ (table **??**). On the other hand, as we saw in figure 5 equation 10 was close to be non-smooth (non-differentiable) in a neighborhood of the solution. That is the reason why neither Newton's method nor the secant algorithm are able to converge even for initial guesses close to the true solution.

If the initial value lies where the slope in figure 5 is almost 0 the method takes a very large step in the first update and the algorithm is unable to find the way back. If on the other hand, the initial value fells in a region where the curve is very steep, the step size in every iteration will be very small and the algorithm will never converge.

A way around would be to include an extra parameter that controls the step size every time σ is updated. However, in this case the number of iterations increases dramatically for small values of ϵ. Thus, it might be advisable to use an algorithm that does not require partial derivatives, such as the bisection method. Applying the bisection method with $\epsilon = 0.01$ the algorithm converges after 24 iterations.

2 9-point Laplacian

This section will extend the implementation of the 5-point Laplacian to a 9-point stencil and discuss how to form the right hand side of the Poisson equation using the correction 3.19 from LeVeque to achieve $\mathcal{O}(h^4)$ convergence in terms of global error (methods to find it can be found in the appendix of LeVeque).

The spacial domain will be unit square defined as $[0, 1] \times [0, 1]$ and discretized to a regular grid with $m \times m$ interior points ($(m + 2) \times (m + 2)$ with boundaries). Dirichlet BC are prescribed everywhere on the boundary, but the method `form_rhs(m,f,u)` will use the "exact" u to calculate them for the sake of simplicity.

2.1 9-point stencil

Visualization of the stencil is shown in figure 3.1b of LeVeque and is achieved in a similar manner as the 5-point stencil, that is by replacing the derivatives with centered finite differences to get the discretization. Since the grid is uniform this leads to equation 3.10 (LV) for the 5-point stencil:

$$\frac{1}{h^2}(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}) = f_{i,j} = \nabla_5^2 u_{i,j} \tag{13}$$

It can be seen that a "cross" like shape is used with the coefficients adding up to 0. As mentioned in the book, this stencil has some drawbacks so let's look at the 9-point one.

The approximation is given by

$$\begin{aligned}\nabla_9^2 u_{i,j} = \frac{1}{6h^2}(&4u_{i-1,j} + 4u_{i+1,j} + 4u_{i,j-1} + 4u_{i,j+1} \\ &+ u_{i-1,j-1} + u_{i-1,j+1} + u_{i+1,j-1} + u_{i+1,j+1} \\ &- 20u_{i,j})\end{aligned} \tag{14}$$

in order to achieve the fourth-order accuracy a clever expansion is presented in LeVeque: the approximation is applied to the true solution and expanded into a Taylor series, with the dominant error term rewritten as $u_{xxxx} + 2u_{xxyy} + u_{yyyy} = \nabla^2(\nabla^2 u) \equiv \nabla^4 u$. For Poisson equation $\nabla^2 u = f$, it is possible to further rewrite $u_{xxxx} + 2u_{xxyy} + u_{yyyy}$ as $\nabla^2 f$. Given these results, the dominant error term can be computed with just the knowledge of the function f. More so, if f is *zero* or a *harmonic* function, then the aforemention term in the local truncation error disappears and the 9-point Laplacian yields a fourth order accurate discretization.

This approach can be extended for arbitary smooth functions f by defining

$$\nabla_9^2 u_{ij} = f(x_i, y_j) + \frac{h^2}{12}\nabla^2 f(x_i, y_j) \tag{15}$$

to cancel out the $\mathcal{O}(h^2)$ terms and end up with $\mathcal{O}(h^4)$ as the local truncation error of this method.

Particularly useful trick for numerical methods when the values of f are only known at the grid points is by using the 5-point Laplacian as follows:

$$\nabla_9^2 u_{ij} = f(x_i, y_j) + \frac{h^2}{12}\nabla_5^2 f(x_i, y_j). \tag{16}$$

Having $\mathcal{O}(h^4)$ out of the way, it is possible to implement the `form_rhs` method by using `poisson9` and `poisson5` function to calculate the respective Laplacians.

2.2 Case problems

Three equations were given to test out the implementation. The first equation is used to make sure that the desired convergence in terms of the global error is achieved.

$$u_{0,exact}(x,y) = \sin(4\pi(x+y)) + \cos(4\pi xy) \tag{17}$$
$$u_{1,exact}(x,y) = x^2 + y^2 \tag{18}$$
$$u_{2,exact}(x,y) = \sin(2\pi|x-y|^{2.5}) \tag{19}$$

Functions u_1 and u_2 will have different rates of convergence of the global error, explanation as to why will be provided later on in this section.

For all the aforementioned functions u, functions f were simply found analytically. One can use Mathematica, Matlab's Symbolic Toolbox or any other CAS to obtain them.

First case

$$u_0(x,y) = \sin(4\pi(x+y)) + \cos(4\pi xy) \tag{20}$$
$$f_0(x,y) = -16\pi^2\left(\cos(4\pi xy)x^2 + \cos(4\pi xy)y^2 + 2\sin(4\pi(x+y))\right) \tag{21}$$

The method `form_rhs(m,f,u)` accepts m as the dimension of the inner point grid $m \times m$, f is the function handle for the RHS of $\nabla^2 u = f$ and finally u is passed in as a function handle to calculate the boundaries of the grid. Note that u is passed in only for simplicity, one could add in an option to provide the boundaries as a matrix instead.

Solution is simply found by using the direct method of solving a system $Au = f$ using Matlab's backslash operator.

Second case

$$u_1(x,y) = x^2 + y^2 \tag{22}$$
$$f_1(x,y) = 4 \tag{23}$$

Third case

$$u_2(x,y) = \sin(2\pi|x-y|^{2.5}) \tag{24}$$
$$f_2(x,y) = 5\pi\sqrt{|x-y|}\,\mathrm{sgn}(x-y)^2\left(3\cos(2\pi|x-y|^{\frac{5}{2}}) - 10\pi|x-y|^{\frac{5}{2}}\sin(2\pi|x-y|^{\frac{5}{2}})\right) \tag{25}$$

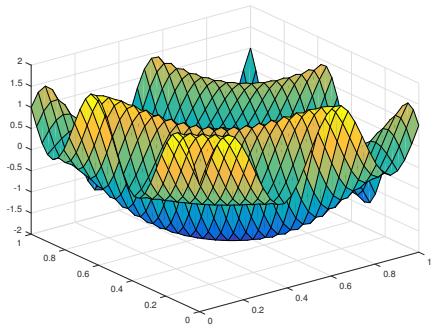

Figure 7: $u_{0,exact}$ on 32×32 grid

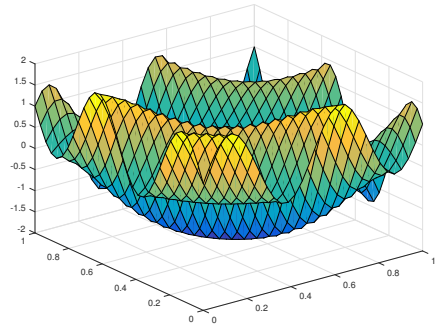

Figure 8: Calculated u using `poisson9 \form_rhs`

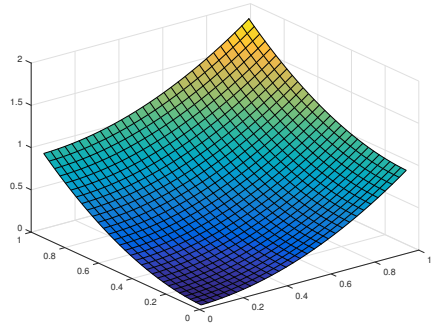

Figure 9: $u_{1,exact}$ on 32×32 grid

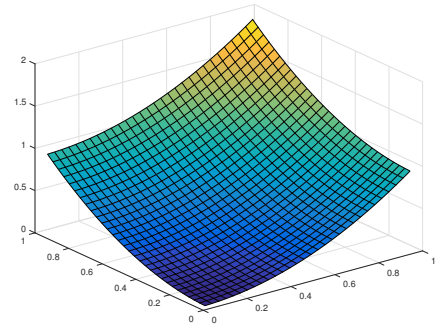

Figure 10: Calculated u using `poisson9 \form_rhs`

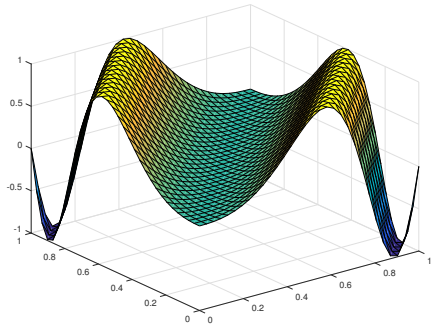

Figure 11: $u_{2,exact}$ on 32×32 grid

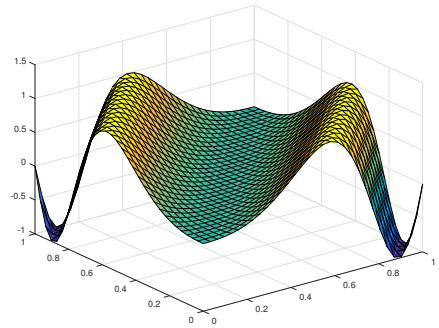

Figure 12: Calculated u using `poisson9 \form_rhs`