Project 3: A semilinear elliptic equation

Estelle Baup and Samuel Bélisle

Spring 2023

Question 1

Take $\alpha > 0$ and $f : \Omega \to \mathbb{R}$. We want to find the weak formulation to the problem of finding u_{n+1} satisfying the problem

$$\begin{cases} -\Delta u_{n+1} + \alpha u_n^2 u_{n+1} = f & \text{in } \Omega, \\ u_{n+1} = 0 & \text{on } \partial\Omega, \end{cases}$$

where $u_n : \Omega \to \mathbb{R}$ is fixed, representing the current iterate, with $u_n = 0$ on $\partial\Omega$.

Weak form

We multiply by a test function $v \in H_0^1(\Omega)$ and integrate over Ω to obtain:

$$\int_\Omega \left(-\Delta u_{n+1} v + \alpha u_n^2 u_{n+1} v\right) = \int_\Omega f v.$$

We use integration by parts to rewrite the left hand side:

$$\begin{aligned} \int_\Omega (-\Delta u_{n+1} v) &= -\int_\Omega \operatorname{div}(\nabla u_{n+1} v) + \int_\Omega \nabla u_{n+1} \nabla v && \text{by integration by part} \\ &= -\oint_{\partial\Omega} v(\nabla u_{n+1} \cdot \vec{n}) + \int_\Omega \nabla u_{n+1} \nabla v && \text{by the divergence theorem} \\ &= \int_\Omega \nabla u_{n+1} \nabla v && \text{because } v \in H_0^1(\Omega), \end{aligned}$$

where we denote $\vec{n}$ the outer normal vector on the boundary $\partial\Omega$.

The weak formulation is:

$$\int_\Omega \nabla u_{n+1} \nabla \phi + \int_\Omega \alpha u_n^2 u_{n+1} \phi = \int_\Omega f\phi \quad \forall \phi \in H_0^1(\Omega). \tag{E1}$$

We have a stiffness integral, a mass integral with the reaction term αu_n^2, as well as a Poisson right hand side.

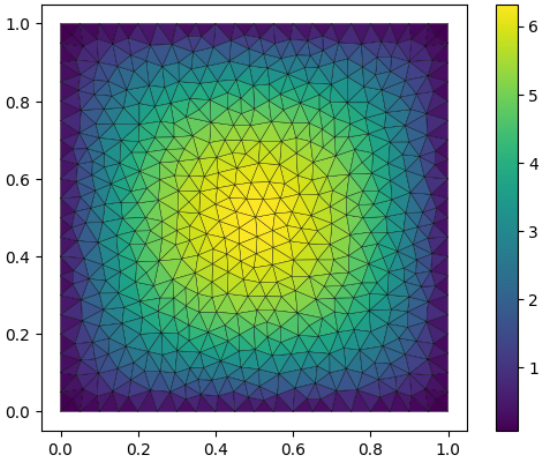
Question 2

We here focus on the fixed-point scheme as presented in the precedent section.

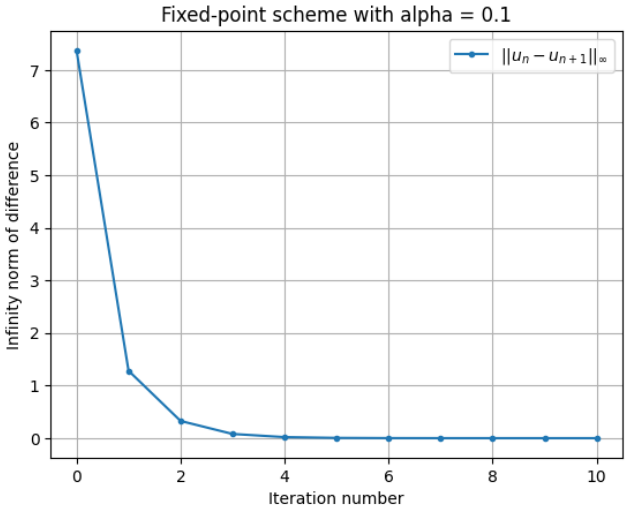
The code for our implementation of this scheme is in the files `integrate.py` and `Project_SEZAM.py`. The scheme is run by the function `fixed_point_method()` of the file `Project_SEZAM.py` , which needs as parameters the value of α, the mesh size, and two arbitrary stropping conditions; We stop when the difference between two iterates is below a given treshold or after a given number of iterations. If that number of iteration treshold is attained before the difference gets low enough, we plot the last iterates. This allowed us to debug the code when it was not working correctly, and can help spot cycling behaviours.

We used a mesh size of $h = 0.05$ to ensure that the mesh contains more than 100 vertices, and the given threshold of 10^{-6}. We also set the maximum of iterations to 20 or 500, chosen to be sure to observe a convergence of our scheme at some step, but avoid letting the program run too much if some code or parameter mistake is made.

For $\alpha = 0.1$, the difference value $||u_n - u_{n+1}||_\infty$ drops below 10^{-6} after 11 iterations. We obtain the figures 1a and 1b below, by running `fixed_point_method(0.1, 10e-6, 20, 0.05)`.



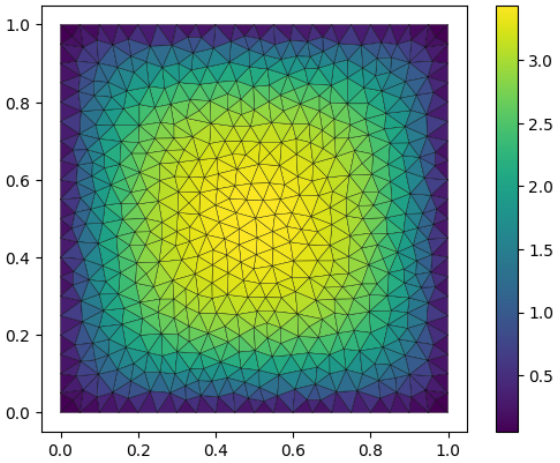
(a) Solution.



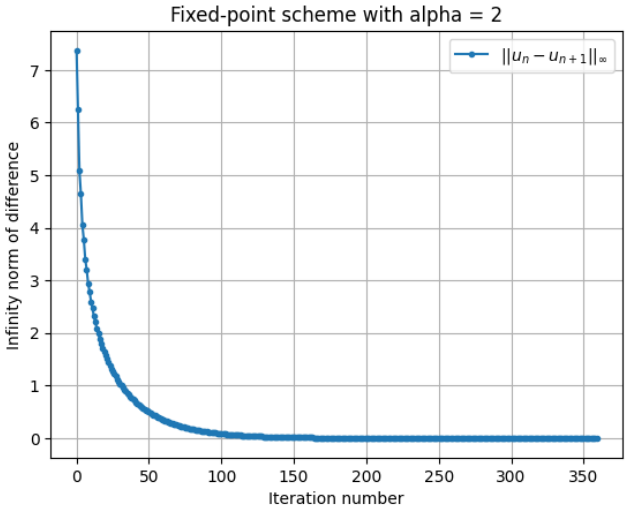
(b) Convergence.

Figure 1: Solution and convergence rate of the fixed-point scheme for $\alpha = 0.1$

For $\alpha = 2$, $||u_n - u_{n+1}||_\infty$ drops below 10^{-6} after 360 iterations, which is much longer. We obtain the figures 2a and 2b by running `fixed_point_method(2, 10e-6, 500, 0.05)`.



(a) Solution.



(b) Convergence.

Figure 2: Solution and convergence rate of the fixed-point scheme for $\alpha = 2$

Question 3

In hope to get a better convergence rate, we implemented the Newton scheme as indicated in the project sheet and below.

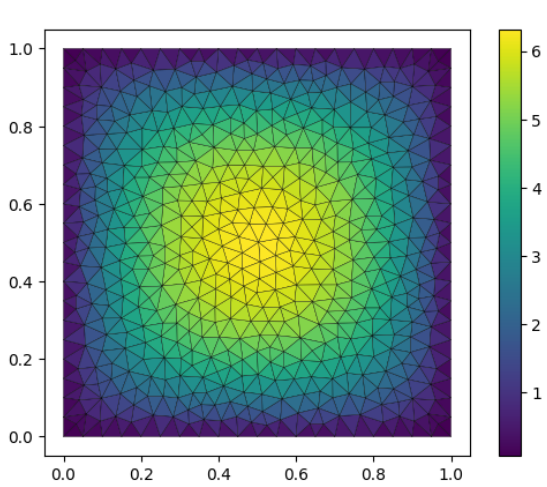
The code for our implementation of is in the same files `integrate.py` and `Project_SEZAM.py`. The function that runs the scheme is `newton_method()`, which takes the same parameters as the function for the fixed-point scheme (cf Question 2). At each iteration, we need to search ∂u_n that satisfies

$$\int_\Omega \nabla\phi \cdot \nabla \partial u_n + 3\alpha u_n^2 \phi \partial u_n \mathrm{d}\Omega = -\int_\Omega \nabla\phi \cdot \nabla u_n + \phi(\alpha u_n^3 - f)\mathrm{d}\Omega \qquad \forall \phi \in H_0^1(\Omega),$$
$$\partial u_n = 0 \qquad \text{on } \partial\Omega.$$

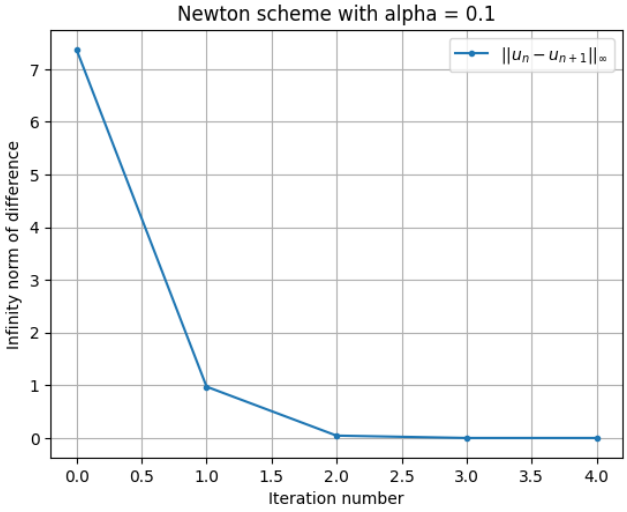
This is a problem with the same stiffness and mass integrals, except for the reaction term that becomes $3\alpha u_n^2$. The main complication is the right hand side.

Then, we seek the solution u as the limit of the recursive sequence $u_{n+1} = u_n + \partial u_n$. The solution ∂u_n is computed with the function `newton_iteration()`.

For $\alpha = 0.1$, we obtain $||u_n - u_{n+1}||_\infty < 10^{-6}$ after 5 iterations, the solution and convergence rate are shown in figure 3a and 3b below, obtained after running `newton_method(0.1, 10e-6, 20, 0.05)`.



(a) Solution.

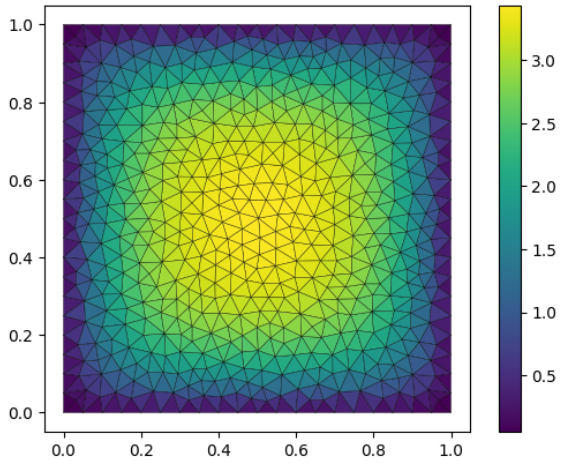


(b) Convergence.

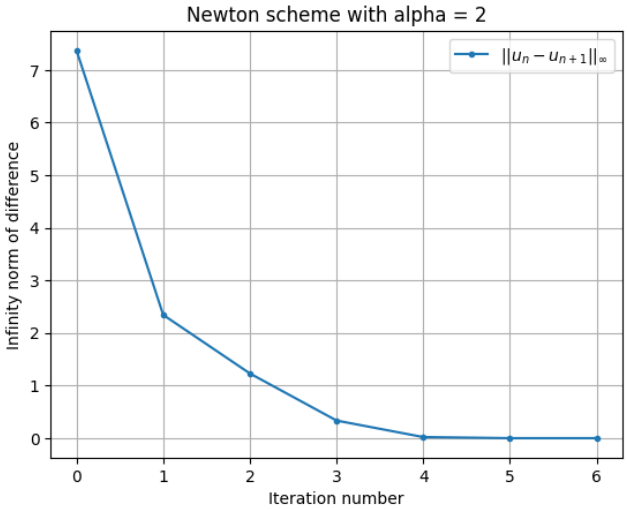
Figure 3: Solution and convergence rate of the Newton scheme for $\alpha = 0.1$

We see that we find the same solution as with the fixed-point method (see figure 1a), and we just needed 5 less iterations with the Newton scheme compared to the fixed-point scheme.

For $\alpha = 2$, we converge in 7 iterations, which is considerably faster than with the fixed-point scheme (we needed 360 iterations to converge with this scheme). We obtain the following results by running `newton_method(2, 10e-6, 20, 0.05)`:



(a) Solution.

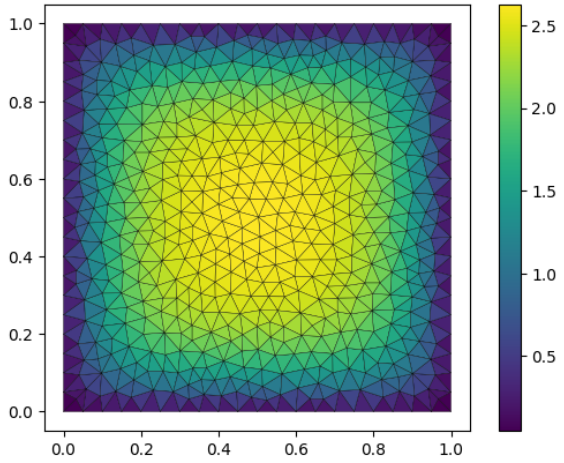


(b) Convergence.

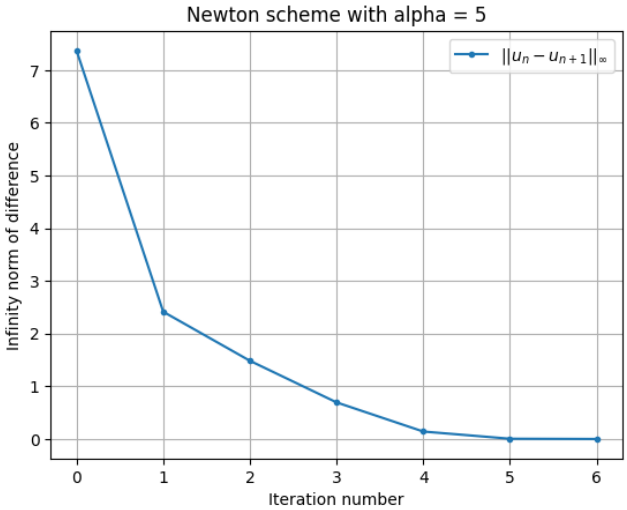
Figure 4: Solution and convergence rate of the Newton scheme for $\alpha = 2$

We also see that the solutions are the same for both scheme (figure 4a above is the same as figure 2a).

For $\alpha = 5$, we also converge in 7 iterations, and obtain the results shown in figures 5a and 5b below (by running `newton_method(5, 10e-6, 20, 0.05)`).



(a) Solution.



(b) Convergence.

Figure 5: Solution and convergence rate of the Newton scheme for $\alpha = 5$

The Newton scheme fares much better than the fixed-point scheme. The number of iterations to obtain $||u_n - u_{n+1}||_\infty < 10^{-6}$ is considerably smaller. (We also ran the scheme for $\alpha = 100$, and found that it converges in 10 iterations.)

In the main function of `Project_SEZAM.py`, there is a bonus call, with a smaller grid, just to show off that the scheme converges quickly, even though our implementation is not optimized for it. For instance, we calculate the mass matrix from scratch at each iteration, although it is always the same.