1 Update Scheme

I haven't found anything useful in the literature yet. I have mainly found things involving gradient decent methods, or different kind of fixed point schemes (which may be worth looking into in general). I have looked at a way of making the adaptive method dependent on the error rather than the iteration count, but haven't been that successful in finding good cutoffs. I started looking at a method which increases λ not in steps but by using the ln function. This seems to work well, see Figures 1, 2. I think there is again ways to improve this, either by trying different things, or by finding the right inspiration in the literature.

```
function res = AdaSolver3(wIn,wOut,wErr, wErrN, wErrVec,lambda, k, j)

    if wErrN > wErr && k > 10
        k = k - 5;
        lambda = lambda*0.7;
    elseif wErrN > wErr && k > 2
        k = k - 2;
        lambda = lambda*0.5;
    elseif wErrN > wErr && k <= 2
        lambda = lambda*0.5;
    elseif wErrN == wErr
        lambda = lambda*0.5;
    elseif wErrN < wErr && k <= 1
        lambda = 0.01;
    elseif wErrN < wErr && k > 2
        lambda = 0.1*log(0.1*k + 1);
    end
    wIn = (1-lambda)*wIn + lambda*wOut;

    if lambda < 0.0001 || wErrN > 10^4
        disp('diverged')
        return
    end

    k = k+1;
    res.k = k;
    res.lambda = lambda;
    res.wErr = wErr;
    res.wIn = wIn;
end
```

Figure 1: New Adaptive Method, k starts at 1, $wErrN$ is the error at this iteration, $wErr$ is at the previous iteration.

1.1 Convergence

The convergence plots are generally very similar and look like Figures 3/ 4. Maybe there is a case for adapting λ to reflect the convergence pattern somehow? Because in the plot it looks like λ drives the error down in the beginning, so that the tolerance is reached sooner, but doesn't change the shape of convergence.However, I am not sure if that is possible, given that the error is quite sensitive to λ even when it's low and slowly changing.

2 The in/outflow problem

I fixed a few mistakes in the code, however it is still not running as nicely. I haven't gotten around to doing everything we suggested trying last week. However, if I set the forward problem

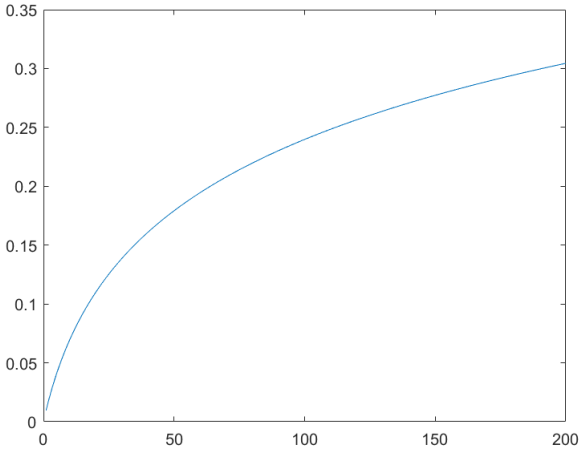


Figure 2: λ vs iterations

(with $w = 0$) as the target, the problem converges within one iteration. I then tried a very small perturbation of this target (without changing the boundaries as we have done in the past). Choosing $\beta = 10^{-1}$, this problem converges to the set error tolerance of 10^{-3}, however not for $\beta = 10^{-2}$. I think this shows that it is working in principle, but that these problems are generally hard. I tried targets further away from the forward problem, and I need to set λ much smaller and it takes way too long to converge (if it does converge). Next I will go the step back to the constant flow on one boundary.

3 Sparse to fine grid

Using 10^{-7} ODE throughout. Testing the exact problem and perturbing it works. There is some gain in iterations. Essentially the fine grid picks up where the sparse grid has left off (e.g. if I run the sparse grid up to a tolerance of 0.1, then the fine grid starts at 0.1 and continues). A time saving can be made because in the exact problem I can run the entire problem (up to the desired optimality tolerance) on the sparse grid. Then the fine grid converges in one iteration. This is faster because each iteration is faster on the sparse grid. Example: For sparse grid choose $N = 22$ and for fine $N = 40$. If we choose to run the sparse grid to an optimality tolerance of 1, this converges within 1 iteration and 2 seconds. We get an output error of 0.0388, which is the same when we interpolate the solution onto the fine grid. It takes 285 iterations and 4000s to converge to an error of 2.8313×10^{-4}.
In comparison, letting the sparse grid run to the optimality tolerance of 10^{-3} (equal to the fine grid), we get that we converge in 357 iterations and 655 seconds. The output error (equal to the interpolated error) is 2.7060×10^{-4}. The fine grid takes one iteration to converge, which is

30 seconds and the final error is 1.0792×10^{-4}. While the second option takes more iteration, it is quite a large saving of time.
However, this does not translate one to one to non exact problems, but I am still in the process of working this out exactly. But the interpolation is not as exact I think because the error of the sparse grid is not equal to the initial error of the fine grid.

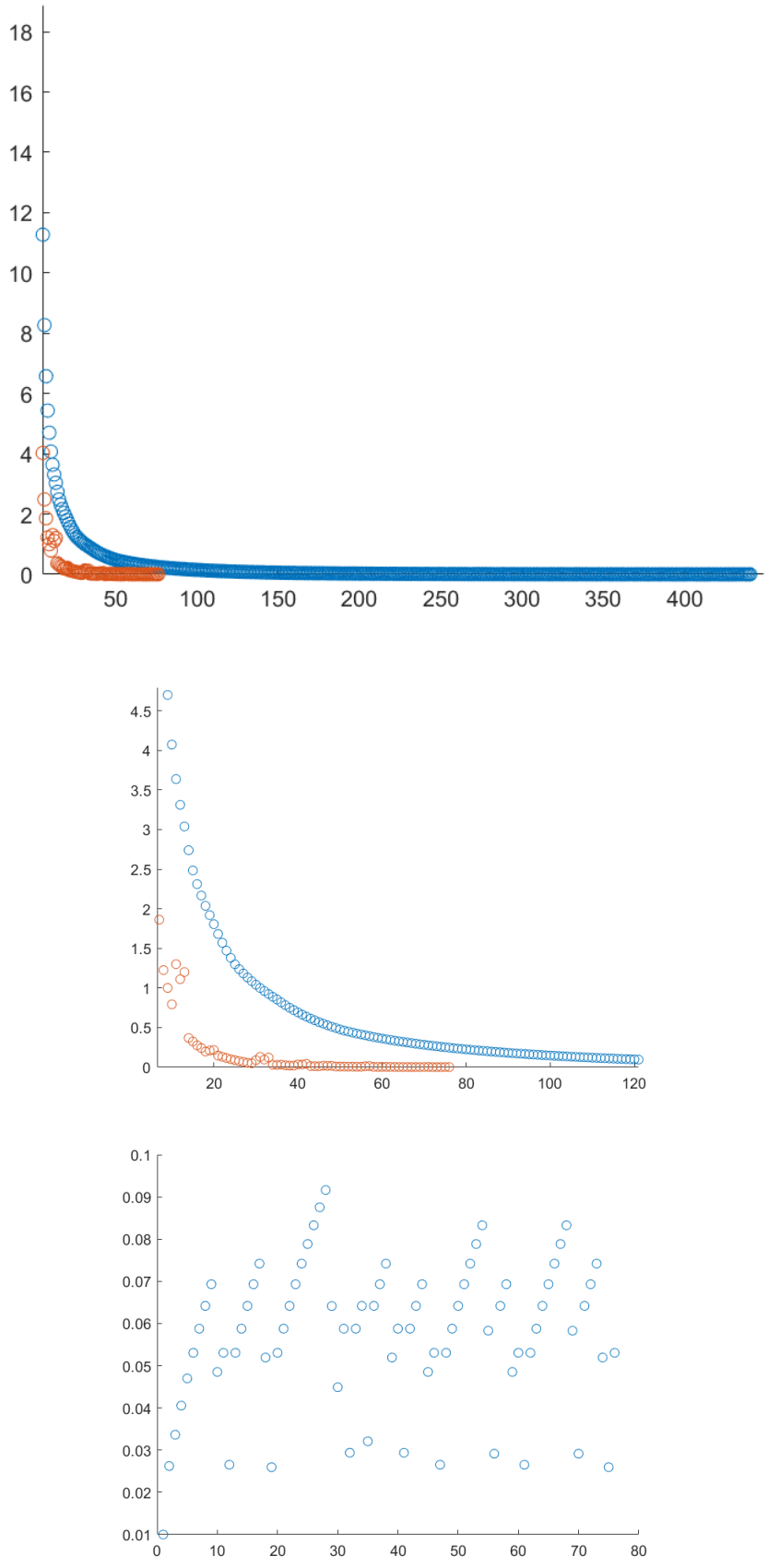
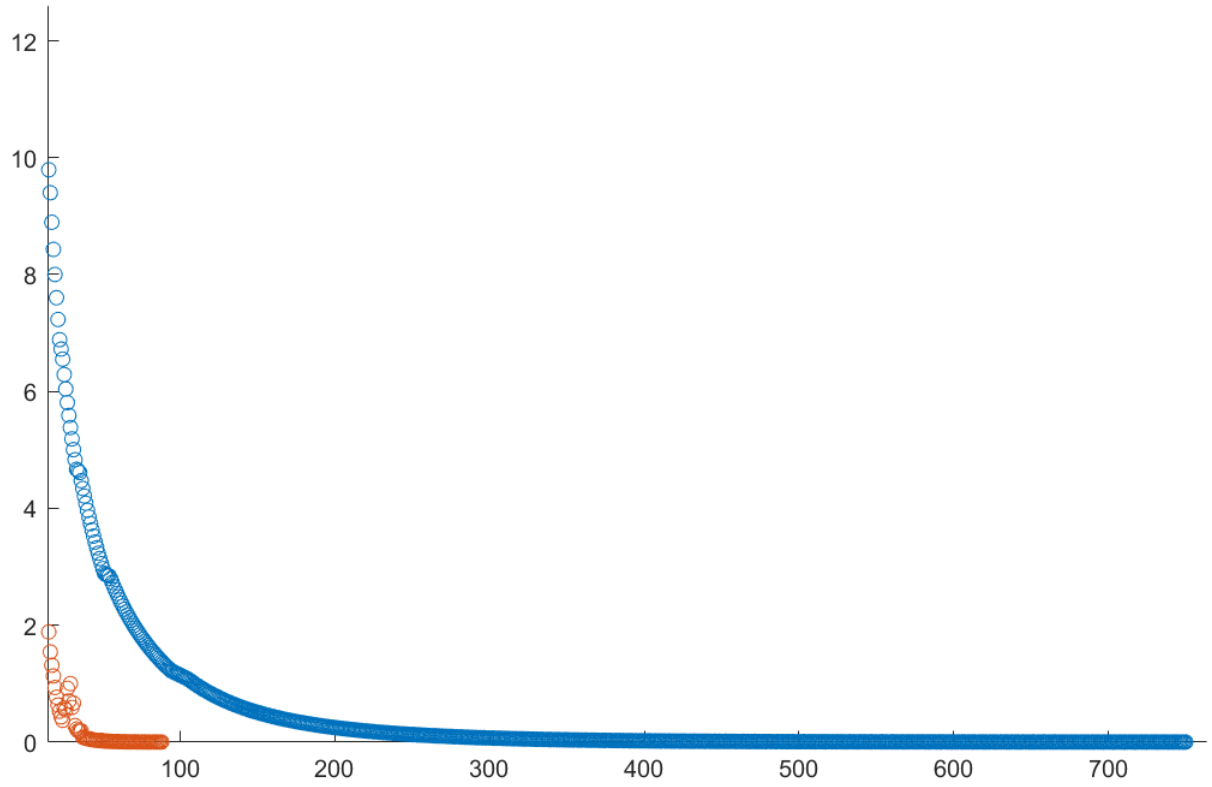
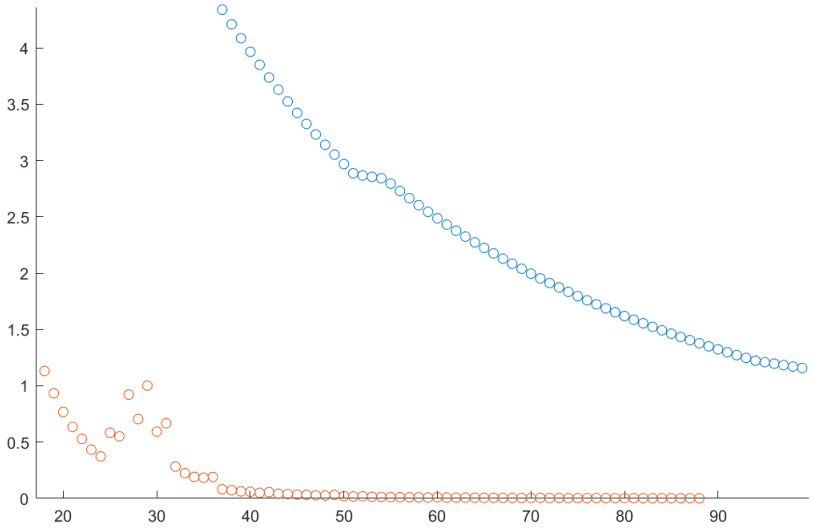
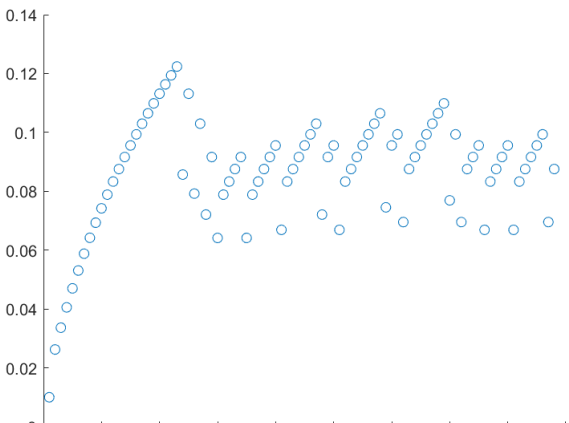


Figure 3: Convergence Plots (blue is λ constant, red is adaptive); change in λ with iteration (figure 3)


12
10
8
6
4
2
0
100
200
300
400
500
600
700


4
3.5
3
2.5
2
1.5
1
0.5
0
20
30
40
50
60
70
80
90


0.14
0.12
0.1
0.08
0.06
0.04
0.02