SOLUTIONS TO ASSIGNMENT 4

Nick Trefethen, 11 March 2020

The six numbers are about about 2.98142, 2.70830, 1.06036, 3437.5, 6.0589, and 0.45809. In the solutions below, please excuse the extensive use of Chebfun. I reached for the tool I know best, but there is no expectation that you had to use Chebfun too.

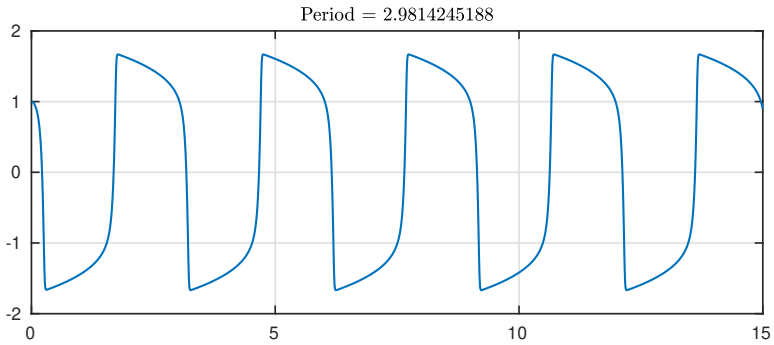
This was big assignment. The whole course through two terms was a major effort, and congratulations on reaching the end! I have been most impressed with people's hard work and creativity, and I am sorry it hasn't been possible to get to know most of you in person.

1. Quartic Van der Pol period

Here's a simple Chebfun solution. The period appears to be around 2.9814245188, quite possibly to all these digits of accuracy.

```
tic, format long
N = chebop(0,15);
N.op = @(u) .05*diff(u,2) - (1-u^4)*diff(u) + u;
N.lbc = [1; 0]; u = N\0; plot(u);
[a,b] = max(u,'local');
Tvec = diff(b), T = Tvec(end-1);
title(sprintf('Period = %14.10f',T))
```

```
Tvec =
   1.773383121623003
   2.981424518806906
   2.981424518801831
   2.981424518795361
   2.981424518794645
```

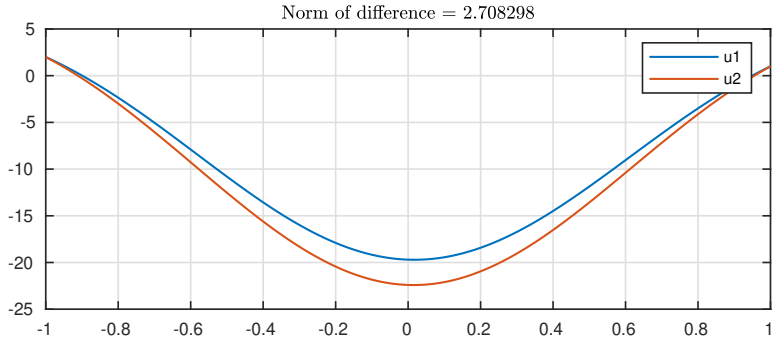


2. Ill-conditioned BVP

I had intended to use coefficients 0.050 and 0.051 on this problem, but accidently wrote 0.50 and 0.51, making it less interesting. Too bad. Again, a simple Chebfun solution. The answer appears to be 2.70830.

```
L = chebop(-1,1);
L.lbc = 2; L.rbc = 1;
L.op = @(x,u) .50*diff(u,2) + x*diff(u) + 2*u; u1 = L\0;
L.op = @(x,u) .51*diff(u,2) + x*diff(u) + 2*u; u2 = L\0;
```

```
plot([u1 u2]), legend('u1','u2'), normdiff = norm(u2-u1,inf);
title(sprintf('Norm of difference = %10.6f',normdiff))
```



3. Quartic Schrodinger equation

This problem is readily solved with Chebyshev spectral methods. We can modify the M-files given in the course or do it with Chebfun, like this:

```
L = chebop(-5,5);
L.op = @(x,u) -diff(u,2) + x^4*u; L.lbc = 0; L.rbc = 0;
e = eigs(L)

e =
   1.060362090483852
   3.799673029801288
   7.455697937986480
  11.644745511377639
  16.261826018850012
  21.238372918235811
```

This suggests that the answer is 1.06036.... Is that accurate, and is it the right eigenvalue? Well it's probably the right eigenvalue, since the quadratic Schrodinger operator is well known to have eigenvalues $1, 3, 5, 7, \ldots$ on the infinite interval. On $[-5, 5]$ the quadratic operator gives via Chebfun

```
L.op = @(x,u) -diff(u,2) + x^2*u; e = eigs(L)

e =
   1.000000000153330
   3.000000007343337
   5.000000168037789
   7.000002442912308
   9.000025274501047
  11.000197435820450
```

Since x^4 looks roughly like x^2 — both potential wells will confine this low eigenvalue to a region approximately $[-1, 1]$ — it looks as if $1.06036\ldots$ should indeed be the right eigenvalue. The fact that the quadratic analogue matches 1 to 9 digits suggests it's very likely that $1.06036\ldots$ is accurate to a number of digits. We could compute the residual like this:

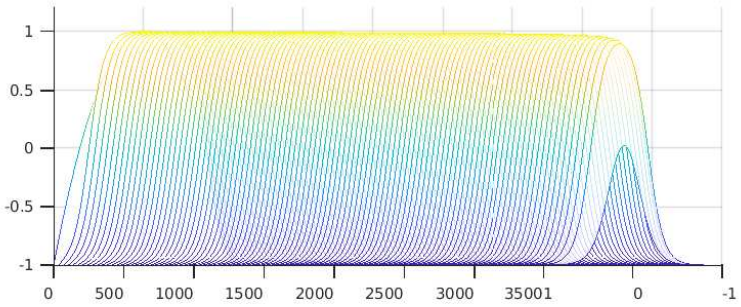
```
[v,lam] = eigs(L,1); norm(L*v-lam*v)

ans = 4.613876682515136e-12
```

which further suggests that there are quite a few digits of accuracy.

4. Allen-Cahn equation

Using codes from the lectures, I would probably adapt `m58_allencahn.m` for this problem. In Chebfun, there's a code called `pde15s` which you can find out about through `chebgui` if you don't already know about it. The critical time emerges as about 3437.5, as this computation shows.

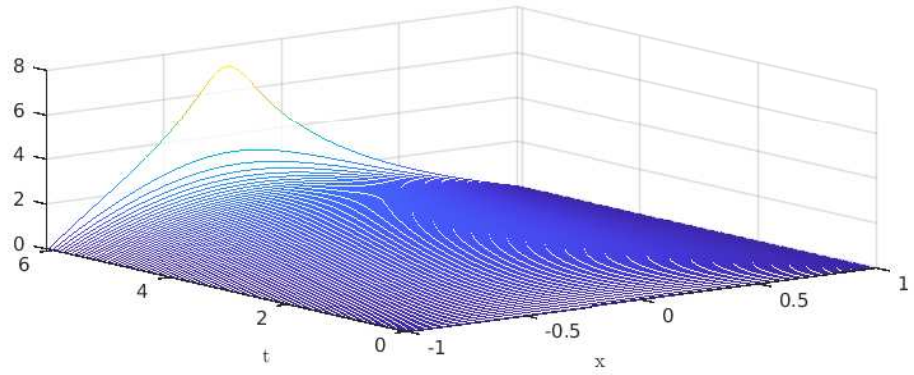
```
t = 0:34.375:3437.6;
pdefun = @(t,x,u) .015*diff(u,2)+u-u.^3;
bc.left = @(t,u) u+1; bc.right = @(t,u) u+1;
x = chebfun(@(x) x); u0 = 1-2.*x.^2;
opts = pdeset('Eps', 1e-6, 'Ylim', [-1,1.1]);
[t, u] = pde15s(pdefun, t, u0, bc, opts);
waterfall(u,t), view(110,0), zlim([-1 1.2])
```



5. Blowup problem

I used pde15s again for this one, again running bisection by hand. I think the critical time is about 6.0589. This is based on the following kind of computation. I varied the value of Eps and the number of time steps and got consistent results.

```
t = linspace(0,6.058,100);
pdefun = @(t,x,u) diff(u,2)+diff(u)+exp(u);
bc.left = 'dirichlet'; bc.right = 'dirichlet';
x = chebfun(@(x) x); u0 = chebfun(0);
opts = pdeset('Eps', 1e-7);
[t, u] = pde15s(pdefun, t, u0, bc, opts);
waterfall(u, t), xlabel('x'), ylabel('t')
max(u(:,end)), axis([-1 1 0 6.1 0 8])

ans = 7.124181944341990
```



6. Heat equation on a cube

The mathematically right method for this problem is separation of variables, which gives a series that converges exponentially. Duncan Martinson did this and found the solution to be 0.45806814580673.

The method I used, on the other hand, was crude numerics — simply discretize by the standard 7-point stencil implemented via `kron`, somewhat as in our codes `m49_leapfrog2D` and `m55_grayscott` but in 3D. I think the final number is around 0.45809. From the raw data, I get this to 3 digits, and then one round of Richardson extrapolation gives two more digits.

```
a = [];
for logJ = 2:6
  J = 2^logJ;
  h = 2/J; s = (h:h:1)';
  I = speye(J-1); D = h^(-2)*toeplitz([-2 1 zeros(1,J-3)]);
  L = kron(I,kron(I,D)) + kron(I,kron(D,I)) + kron(D,kron(I,I));
  rhs = -h^(-2)*[ones((J-1)^2,1); zeros((J-1)^3-(J-1)^2,1)];
  u = L\rhs;
  JJ = (J-1)^2*(J/4 - 1);
  slice = u(JJ+1:JJ+(J-1)^2);   % 1/4 of the way from the hot face
  val = max(slice)              % value at middle of this slice
  a = [a; val];
end
b = a(2:end) + diff(a)/3        % Richardson extrapolation


val = 0.435574229691877
val = 0.450454154658691
val = 0.455989943700141
val = 0.457549816063755
val = 0.457951759981428
b =
   0.455414129647629
   0.457835206713957
   0.458069773518293
   0.458085741287319
```

Most of the following number comes from the case $J = 64$ in the very last calculation.

```
Total_time_for_this_solution_sheet = toc


Total_time_for_this_solution_sheet = 21.071973000000000
```