Integral Equations and Fast Algorithms (CS 598AK)

Homework Set 3

Due: October 3, 2013 · Out: September 19, 2013

Problem 0: Propose a project

Write about half a page on the project you would like to do for this class. The project consists of the following components:

- A 20-minute oral presentation. (+5-ish minutes for questions) This will constitute 40% of the grade for the final project.

  We'll set aside about the last three lectures for this:

  - November 21
  - (No class on November 26 and 28 because of Thanksgiving break.)
  - December 3
  - December 5
  - (No class on December 8–I'll be at a workshop in Banff.)

- An implementation component. (0–40% of project your grade)
- A written report. (20–60% of your project grade)

  At a minimum, this report will describe a number of numerical experiments that are enabled by your implementation and how they are to be interpreted.

The relative weighting of implementation and written report is not fixed. If you decide to tackle an ambitious implementation project, you can turn in a less ambitious report that amounts to documentation for your code. Note that it is acceptable (but not necessarily recommended) to only write a report and not implement anything.

It's OK (even encouraged) to choose something for your project that fits within your research. The project has to be small and independent enough that you can meaningfully describe what's going on within your 20 minute talk and share your code in a way that your classmates can play around with your code.

Make sure you answer at least the following answers on your project:

- Write a few sentences on the big picture surrounding the project. What problem does it solve? Where does this problem occur?
- Write a few sentences that survey the literature around your topic. Even if you pick one of the papers from the class web page, try to figure out what new developments have happened in that area since that paper was written.
- Why do you care? Why is this an interesting project? What's compelling about it?
- What date would you like to present?

- What is your desired weighting of report and implementation?
- Describe, as precisely as you can, what you would like to implement. What is the input? What is the computed output? How accurate is the output? What is the asymptotic cost of the computation?
- Describe, as precisely as you can, what your report will contain. What results will you be discussing?

Hints: If you're lost as to what to propose for a project, take a look at the list of papers on the class web page.

This homework set walks you through the process of building a solver for linear boundary value problems of the type

$$u'' + p(x)u' + q(x)u = r(x), \quad u(a) = u_a, \quad u(b) = u_b \tag{1}$$

that uses a second-kind Fredholm integral equation.

Problem 1: Derive a second-kind Fredholm integral equation for the BVP

Derive a second-kind Fredholm integral equation equivalent to (1).

Proceed as follows:

a) Let $\phi := u''$. Find an expression for $u(x)$ using only ϕ and the left boundary values $u(a)$ and $u'(a)$. Call this equality (A).

Hint: This step is exactly the same as what we did for the conversion of the initial value problem to a Volterra IE.

b) Find an expression for $u(x)$ using only ϕ and the right boundary values $u(b)$ and $u'(b)$. Call this equality (B).

The two relations that you have found are not really helpful yet, because they involve $u'(a)$ and $u'(b)$, neither of which we are given as part of (1).

c) To help eliminate the derivative terms, write down another equality (C) that relates $u'(a)$ and $u'(b)$ using ϕ.

Hint: $\int \phi = u'$.

d) Find the linear function that is 1 at a and 0 at b. Call that function τ.

Combine the identities (A), (B) and (C) according to

$$\tau(x)A + (1 - \tau(x))B + F(x)C, \tag{2}$$

i.e. in such a way that (A) is 'felt' at a and (B) is 'felt' at b. Find $F(x)$ so that (C) cancels the derivative terms. Write out an expression for $F(x)$.

Rearrange (2) to isolate $u(x) = \cdots$ on the left hand side.

Hints:

- Write τ in terms of x, a, and b. Realize that the factors in front of $u'(a)$ and $u'(b)$ differ only in sign.
- You're shooting for

$$u(x) = \tau(x)u(a) + (1-\tau(x))u(b) + \frac{1}{b-a}\left((b-x)\int_a^x \phi(z)(a-z)dz + (x-a)\int_b^x \phi(z)(b-z)dz\right). \quad (3)$$

e) Differentiate (3) to gain an expression for u'.

f) Plug the expressions for u, u' and $u'' = \phi$ into (1) to obtain a second-kind integral equation. Write down expressions for the kernel and the right-hand side.

Turn in your answers to the questions above in a PDF `answer.pdf` in the root directory of your `git` repository.

Problem 2: Check solvability

Answer the following questions by referring to theorems from the lecture.

a) Does the Fredholm Alternative apply to this integral equation? Why?

Hint: How 'well-behaved' is the kernel?

b) Write out the statement of the Fredholm Alternative specifically for this problem.

Turn in your answers to the questions above in a PDF `answer.pdf` in the root directory of your `git` repository.

Problem 3: Build a solver

a) Complete the following function:

```
def apply_fredholm_kernel(mesh, kernel, density):
    r"""Return a vector *F* corresponding to

    .. math::

        F(x) = \int_a^b k(x,z) \phi(z) dz

    evaluated at all points of *mesh* using the
    trapezoidal rule.

    :arg mesh: A 1D array of nodes in the interval :math:`[a, b]`,
        with the first equal to :math:`a` and the last equal to :math:`b`.
    :arg kernel: two-argument vectorized callable ``kernel(tgt, source)``
        that evaluates
    :arg density: Values of the density at the nodes of *mesh*.
    """

    # insert code here
```

Submit this code in `bvp.py`.

b) Complete the following function:

```
def solve_bvp(mesh, p, q, r, ua, ub):
    r"""Solve the boundary value problem

    .. math::

        u''+p(x)u'+q(x)u=r(x),\quad u(a) = u_a,\quad u(b) =u_b

    on *mesh*. Return a vector corresponding to the solution *u*.
    Uses :func:`apply_fredholm_kernel`.

    :arg mesh: A 1D array of nodes in the interval :math:`[a, b]`,
        with the first equal to :math:`a` and the last equal to :math:`b`.
    :arg p, q, r: Functions that accept a vector *x* and
      evaluate the functions $p$, $q$, $r$ at the nodes in *x*.
    """

    # insert code here
```

Submit this code in `bvp.py`.

To actually solve the BVP, perform the following steps:

1. Evaluate the right-hand b side on `mesh`.
2. Use GMRES to solve $(I + A)\phi = b$.
   Evaluate $A\phi$ by a call to `apply_fredholm_kernel`.
   Hint: `scipy.sparse.linalg.LinearOperator`, `scipy.sparse.linalg.gmres`. Make sure to check the `info` return value from GMRES.
3. Recover the solution u from ϕ using (3).
   This may be easiest to do by yet again calling `apply_fredholm_kernel` with a different `kernel` argument.

c) Test your code on the following boundary value problems on the interval $(-4, 5)$:

1. `p(x)=0`
   `q(x)=0`
   `r(x)=10*np.exp(x)*np.cos(5*x)-24*np.exp(x)*np.sin(5*x)`
   Correct solution: `u(x)=np.sin(5*x)*np.exp(x)` (also use to obtain boundary data $u(a)$ and $u(b)$)
2. `p(x)=0`
   `q(x)=1/(x+7)`
   `r(x)=np.sin(5*x**2)/(x+7)-100*x**2*np.sin(5*x**2)+10*np.cos(5*x**2)`
   Correct solution: `u(x)=np.sin(5*x**2)` (also use to obtain boundary data $u(a)$ and $u(b)$)
3. `p(x)=np.cos(x)`
   `q(x)=0`

```
r(x)=( - 100*x**2*np.sin(5*x**2)
+ 10*x*np.cos(x)*np.cos(5*x**2)+10*np.cos(5*x**2) )
```

Correct solution: `u(x)=np.sin(5*x**2)` (also use to obtain boundary data $u(a)$ and $u(b)$)

4.
```
p(x)=np.cos(x)
q(x)=1/(x+7)
r(x)=(np.sin(5*x**2)/(x+7)
- 100*x**2*np.sin(5*x**2)
+ 10*x*np.cos(x)*np.cos(5*x**2)+10*np.cos(5*x**2))
```

   Correct solution: `u(x)=np.sin(5*x**2)` (also use to obtain boundary data $u(a)$ and $u(b)$)

Submit `test_bvp.py`, which uses `solve_bvp` to compute an approximate `u` for two different mesh widths h, computes the L^2 error, and estimates the order of accuracy from the two results.

What order of accuracy are you expecting? What order are you observing? (Answer this in a comment in the `test_bvp.py` source file.)

d) Count the number of applications of the discretized operator that GMRES requires to converge. Make your code output this information.

Answer the following questions:

- How does this number of iterations depend on the number of discretization points n?
- How does the number of iterations vary if you request a lower GMRES tolerance? (see also the hints below)
- What does the GMRES residual say about the accuracy with which you solved the boundary value problem?

(Answer this in a comment in the `test_bvp.py` source file.)

Hints:

- Make sure you don't destroy the full order of convergence of the trapezoidal rule. Pay special attention at the kernel 'diagonal' $k(x, x)$. (You might have to use rather many points (up to 6000 in my tests) in the last (most complicated) test problem to see the impact of this.)

  Also note that doing this right may require that you change the interface between your function `apply_fredholm_kernel` and its argument `kernel`.

- `scipy.sparse.linalg.gmres` has a default "tolerance" value `tol` of 10^{-5}. You will likely need to decrease this value.

- This is, once again, an $O(n^2)$ algorithm. We'll fix that soon.