Integral Equations and Fast Algorithms (CS 598AK)

Homework Set 2

Due: September 19, 2013 · Out: September 5, 2013

Problem 1: Solve an ODE initial value problem using a Volterra IE

Consider the initial value problem

$$\frac{d^2y}{dx^2} + u(x)\frac{dy}{dx} + v(x)y = 0, \qquad y(a) = y_0, \qquad y'(a) = y'_0. \tag{1}$$

a) Convert (1) into a Volterra integral equation by the method we have discussed in class. Submit a PDF `problem-1/part-1a.pdf` with your answer and a brief derivation.

b) Complete the following function template:

```
def solve_second_kind_volterra(a, b, n, kernel, rhs):
    r"""Returns a tuple *(mesh, soln)* of *n*-vectors such that
    *soln* represents point values at the points defined by
    *mesh* (which lie in :math:`[a,b]`) of a function :math:`f`
    solving the integral equation

    .. math::

        f(x) + \int_a^x \text{kernel}(x,y) f(y) dy = \text{rhs}(x)

    The integral equation is solved using the trapezoidal rule.

    :arg a: the left-hand side of the solution interval
    :arg b: the right-hand side of the solution interval
    :arg n: the number of equispaced points to use in [a,b]
    :arg kernel: a callable taking two vector arguments *x* and
        *y* representing source and target coordinates
    :arg rhs: a callable taking one vector argument *x*
        representing target coordinates
    """

    # insert code here
```

Submit this code in `problem-1/volterra.py`.

c) Complete the following function template:

```
def solve_second_order_ivp(a, b, n, u, v, y0, y0p):
    r"""Returns a tuple *(mesh, soln)* of *n*-vectors such that
    *soln* represents point values at the points defined by
    *mesh* (which lie in :math:`[a,b]`) of a function :math:`f`
    solving the initial value problem:
```

```
.. math::

    \frac{d^2 y}{dx^2} + u(x) \frac{dy}{dx} + v(x)y = 0,
    \qquad
    y(a)=\text{y0},
    \qquad
    y'(a)=\text{y0p}.

The ODE is solved using the corresponding Volterra integral
equation.

:arg a: the left-hand side of the solution interval
:arg b: the right-hand side of the solution interval
:arg n: the number of equispaced points to use in [a,b]
:arg u: a single-argument callable that evaluates the
  coefficient function *u*
:arg v: a single-argument callable that evaluates the
  coefficient function *v*
"""

# insert code here
```

Also submit this code in `problem-1/volterra.py`. Your function `solve_second_order_ode` should call `solve_second_kind_volterra` (i.e. be a 'wrapper' around it).

d) Test your code on the ordinary differential equation defining the Bessel functions[1] J_α and Y_α:

$$x^2 \frac{d^2y}{dx^2} + x\frac{dy}{dx} + (x^2 - \alpha^2)y = 0$$

Divide the ODE by the leading coefficient (x^2) to bring it into the form of (1). To avoid the singular behavior of the ODE at $x = 0$, we will test on the interval $[a, b] = [1, 20]$.

Obtain initial values for $J_1(1)$ using `scipy.special.jv`[2] and `scipy.special.jvp`.

Note that the Volterra integral equation will compute the second derivative of the solution, so you will have to 'post-process' the solution by quadrature once more. For a variety of grid spacings h, compute an approximate L^2 error in your approximation of J_1 on $[1, 10]$.

What order of convergence do you observe? What order would you expect?

Use this procedure to estimate the order:

We assume that the error depends on the mesh spacings h as $E(h) \approx Ch^p$ for some unknown power p. Taking the log of $(h, E(h))$ turns this into a linear function of p:

$$E(h) \approx Ch^p \quad \Leftrightarrow \quad \log E(h) \approx \log(C) + p \log(h).$$

[1] `https://en.wikipedia.org/wiki/Bessel_function`
[2] `http://docs.scipy.org/doc/scipy/reference/special.html`

You can now either do a least-squares fit for $\log C$ and p from a few data points $(h, E(h))$ (more accurate, more robust), or you can use just two grid sizes h_1 and h_2, and estimate (less accurate, less robust)

$$p \approx \frac{\log(E(h_2)/E(h_1))}{\log(h_2/h_1)}.$$

This is called the *empirical order of convergence* or *EOC*.

Submit a driver file `problem-1/test.py` that performs this test for two counts `n1` and `n2` of equispaced mesh points given on the command line, outputs the errors and the empirical order of convergence.

Hints:

- Feel free to use the code at `https://gist.github.com/inducer/6439136`.
- You can test $J_1''(x)$ (the result of the integral equation solve) individually–just use the Bessel ODE to compute the value of J_1'' from J_1' and J_1, which you can get from `scipy`.

Comments:

- While this *works* as an ODE solver, it's a bit impractical because of its $O(n^2)$ run time. (Realize that that's fixable–a simple blocking scheme will accomplish that here, for example.)
- For an example of a more practical method of solving ODEs using integral equations, see the article on 'spectral deferred correction' at this DOI: doi: 10.1023/A:1022338906936.