Chem/Stat3240: Homework 5b
Matlab

October 2, 2015

4a Modify your function `ellipsePerim` from homework 5a to create a function `ellipsePerim1` that calls a nested function `P_inner1` rather than the subfunction `P_inner0`. The nested function will have one input argument for the number of vertices.

4b Next create a function `ellipsePerim2` that calls a subfunction `P_inner2` rather than the function `P_inner0`. The function `P_inner2` works by computing the first quadrant portion of the polygon perimeter and then multiplying the result by 4. This takes advantage of the symmetry in the perimeter of an ellipse. Make sure the set polygon vertices you use for the first quadrant calculations include $\theta = 0$ and $\theta = \pi/2$, as the endpoints so there will be $n + 1$ vertices in the first quadrant where n is the input number of points. This should be a easy code modification.

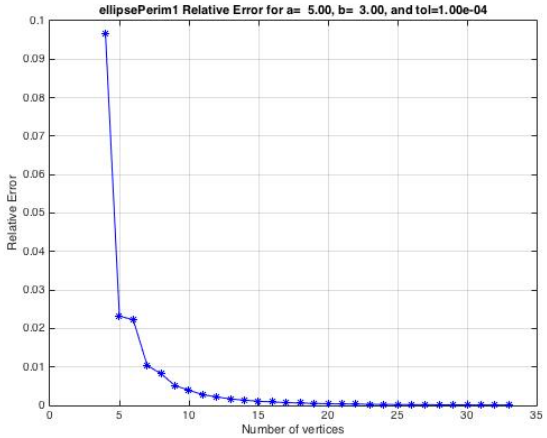
4c Finally create a function `ellipsePerim3` that calls a subfunction `P_inner3` rather than the function `P_inner1` that only uses vectorized operations rather than a looping structure. This can be done in just 5 lines of code.

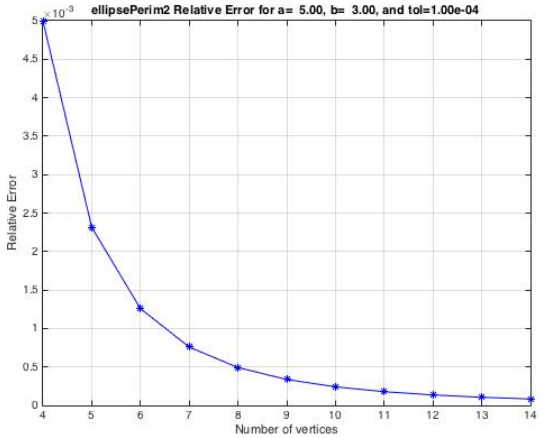
4d Modify your function `ellipsePerim2` to create a function `ellipsePerim4` that calls a subfunction `P_inner4` instead of `P_inner2`. The function `P_inner4` will incorporate the array-based programming forms used in `P_inner3` with computation of vertices just in the first quadrant of `P_inner2` .

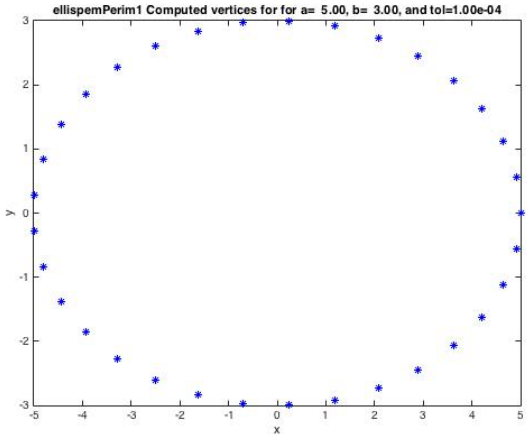
For testing the subfunctions `P_inner2`, `P_inner3`, and `P_inner4`, you will have to put the subfunction into a separate function file so that its

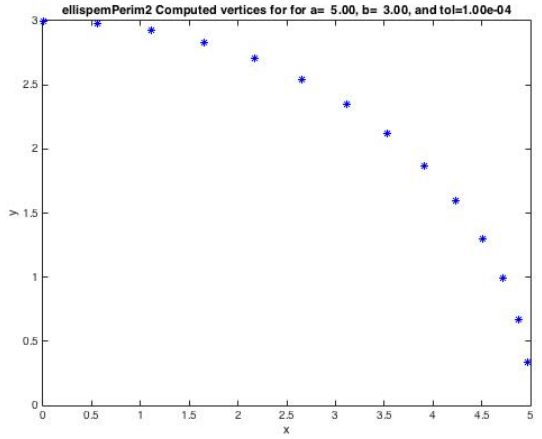
visible to the test.

Run your `ellispePerim1` and `ellipsePerim2` functions, setting the `platsOn` input to `'on'` and duplicate the following plots. Note that `ellipsePerim2` converges to the `tol` in less than half the iterations of ellipsePerim1.









Submit your code file for the functions `ellipsePerim1`, `ellipsePerim2`, and `ellipsePerim3` to the course collab site as well as to Cody. Submit the pdfs of the plots shown above to the collab site as well. Include `P_inner` and your testing frameworks for each version of `ellipsePerim` in the code you submit to the collab site.

5 Write a function `perimPerform(ellipsePerim)` (including the func-

tion specification) that takes an input function handle `ellipsePerim` for computing the ellipse perimeter and creates a table (see below) of computed ellipse perimeters for a logarithmically scaled sequence of input tolerances. See the example script Eg5_2B in your text for how this might be done. In your function, set the ellipse parameters to $a = 5$ and $b = 3$. The function will print the name of the function handle input (see `func2str`) and and the time it takes to generate the table for that function (see `tic` and `toc`).

Use the function `perimPerform` to test the performance for `ellipsePerim1`, `ellipsePerim2`, and `ellipsePerim3`. Your output for the three functions should look similar to the following:

```
>> perimPerform(@ellipsePerim1)


Performance of function ellipsePerim1

  tolerance       Perimeter Estimate
-----------------------------------------
1.00e-01      23.323807579381
1.00e-02      24.883241220330
1.00e-03      25.363300133500
1.00e-04      25.488457810788
1.00e-05      25.518430291682
1.00e-06      25.525132671882
1.00e-07      25.526591355426
1.00e-08      25.526910922335
1.00e-09      25.526979898859
1.00e-10      25.526994775574
1.00e-11      25.526997982457
1.00e-12      25.526998673139
Elapsed time =  19.870542
>> perimPerform(@ellipsePerim2)


Performance of function ellipsePerim2
```

```
  tolerance       Perimeter Estimate
-----------------------------------------
1.00e-01      25.363300133500
1.00e-02      25.363300133500
1.00e-03      25.473473599499
1.00e-04      25.513611228520
1.00e-05      25.523651559652
1.00e-06      25.526269872594
1.00e-07      25.526838683684
1.00e-08      25.526964160743
1.00e-09      25.526991349708
1.00e-10      25.526997243934
1.00e-11      25.526998514089
1.00e-12      25.526998787878
Elapsed time =   2.849632
>> perimPerform(@ellipsePerim3)


Performance of function ellipsePerim3

  tolerance       Perimeter Estimate
-----------------------------------------
1.00e-01      23.323807579381
1.00e-02      24.883241220330
1.00e-03      25.363300133500
1.00e-04      25.488457810788
1.00e-05      25.518430291682
1.00e-06      25.525132671882
1.00e-07      25.526591355426
1.00e-08      25.526910922335
1.00e-09      25.526979898859
1.00e-10      25.526994775574
1.00e-11      25.526997982712
1.00e-12      25.526998672600
Elapsed time =   6.514692
>> perimPerform(@ellipsePerim4)


Performance of function ellipsePerim4
```

```
  tolerance       Perimeter Estimate
-----------------------------------------
1.00e-01      25.363300133500
1.00e-02      25.363300133500
1.00e-03      25.473473599499
1.00e-04      25.513611228520
1.00e-05      25.523651559652
1.00e-06      25.526269872594
1.00e-07      25.526838683684
1.00e-08      25.526964160743
1.00e-09      25.526991349708
1.00e-10      25.526997243934
1.00e-11      25.526998514089
1.00e-12      25.526998788032
Elapsed time =   1.295668
>>
```

Note that the speedup from combining the array-based programming with the savings of computing vertices just across the first quadrant, we speed up the code by more than an order-of-mangnitude.

Cut and past the output of `perimPerform` for each of the cases above into the bottom of the file `perimPerform.m` as a block of comments and submit `perimPerform.m` to the course collab site.