ENGN 2020:

Solutions to Midterm #2

Brown University School of Engineering

Problem 1

Part a. A vertex on the graph is the current state of the board, which contains the contents of each cell as well as whose turn is next; this is the minimum information that should be stored in the vertex. The adjacency method just provides the next possible states of the game: the number of children should be equal to the number of open cells, since the active player can put a mark in any of these. Example code is below for this is contained after part b.

Part b. We are just searching for rows, columns, or diagonals that sum to +3 or -3, then reporting the status to the user based on what we find. This is implemented in the `get_status` method below.

File attached here. 

```
import numpy as np

class Vertex:
    """Represents the current state of the tic-tac-toe game.
    Initialize with a 3x3 array with the board state, and also
    whose turn it is. In both cases, 1:'X', -1:'O', 0: empty"""
    def __init__(self, board, turn):
        self.board = np.asarray(board, dtype=int) #FIXME check speed!
        self.turn = turn
        self.decoder = {0:'-', 1:'X', -1:'O'}
    def __repr__(self):
        s = "{}'s turn:\n".format(self.decoder[self.turn])
        for row in self.board:
            s += ''.join([self.decoder[_] for _ in row])
            s += '\n'
        return s
    def get_children(self):
        """Returns possible moves."""
        children = []
        empty_spaces = np.argwhere(self.board == 0)
        for empty_space in empty_spaces:
            indices = tuple(empty_space)
            b = np.array(self.board)
            b[indices] = self.turn
            children.append(Vertex(b, -self.turn))
        return children
    def get_status(self):
        """Returns current status of game: 'X wins', 'O wins', 'tie', or
        'in progress'."""
        # Sums is all possible sums: horizontal, vertical, and diagonal:
        b = self.board
        sums = np.concatenate([b.sum(axis=0), b.sum(axis=1), [np.trace(b)],
                               [np.trace(np.fliplr(b))]])
        if (3 in sums) and (-3 in sums):
            raise RuntimeError("Invalid board state -- both can't win.")
        if 3 in sums:
            return 'X wins'
        if -3 in sums:
```

```
            return 'O wins'
        if np.count_nonzero(b) == 9:
            return 'tie'
        return 'in progress'
```

Part c. Here, our assignment is not to come up with the "best" strategy in tic-tac-toe, but to use a fairly dumb strategy of just taking the highest probability given all future outcomes. This, of course, could be refined in future implementations. The answers to the four cases, are (in order):

```
X's turn:
O-O
-XX
X-O
Winning outcomes: 1. Total outcomes: 5.
**************************************************
O's turn:
--O
-X-
---
Winning outcomes: 1312. Total outcomes: 3468.
**************************************************
X's turn:
X--
OO-
XOX
Winning outcomes: 4. Total outcomes: 6.
**************************************************
O's turn:
X--
-O-
---
Winning outcomes: 792. Total outcomes: 3198.
```

Example code to implement this is below. File attached here.

```
import numpy as np
from vertex import Vertex

def get_move_probabilities(vertex):
    """Returns all possible moves and the probability associated
    with that move."""
    possible_moves = vertex.get_children()
    moves = []
    for possible_move in possible_moves:
        outcomes = {'X wins': 0, 'O wins': 0, 'tie': 0}
        q = [possible_move]
        while len(q) > 0:
```

```
            v = q.pop(0)
            status = v.get_status()
            if status == 'in progress':
                q.extend(v.get_children())
            else:
                outcomes[status] += 1
        moves.append([possible_move, outcomes])
    return moves

def computer_move(vertex, verbose=False):
    """Given a vertex, selects the best next move, on a purely frequentist
    basis."""
    moves = get_move_probabilities(vertex)
    if vertex.turn == 1:  # X's turn
        wins = np.array([move[1]['X wins'] for move in moves])
    elif vertex.turn == -1:  # O's turn
        wins = np.array([move[1]['O wins'] for move in moves])
    totals = np.array([move[1]['X wins'] + move[1]['O wins'] + move[1]['tie']
                       for move in moves])
    ratios = wins / totals
    choice = np.argmax(ratios)
    move = moves[choice]
    if verbose:
        print(move[0])
        print('Winning outcomes: {}. Total outcomes: {}.'.format(
           wins[choice], totals[choice]))
    return move[0]

if __name__ == '__main__':
    cases = [Vertex([[0, 0, -1], [0, 1, 1], [1, 0, -1]], -1),
             Vertex([[0, 0, -1], [0, 0, 0], [0, 0, 0]], 1),
             Vertex([[1, 0, 0], [-1, 0, 0], [1, -1, 1]], -1),
             Vertex([[0, 0, 0], [0, -1, 0], [0, 0, 0]], 1)]
    for case in cases:
        computer_move(case, verbose=True)
        print('*' * 50)
```

Part d. What's left is just to assemble the pieces from above into a text-based game. An example follows. File attached here.

```
import numpy as np
from move import computer_move
from vertex import Vertex

def game():
    """Play tic tac toe!"""
    if np.random.rand() < 0.5:
        print('Computer (O) goes first.')
        ttt = Vertex(np.zeros((3, 3)), turn=-1)
    else:
        print('Human (X) goes first.')
        ttt = Vertex(np.zeros((3, 3)), turn=1)
    game_over = False
    while not game_over:
```

```
        print('-'*20)
        print(ttt)
        if ttt.turn == 1:
            move = input('Enter move:')
            move = np.array(move.split(','), dtype=int)
            if ttt.board[move[0], move[1]] != 0:
                raise RuntimeError('invalid move!')
            _ = np.array(ttt.board)
            _[move[0], move[1]] = 1
            ttt = Vertex(_, turn=-1)
        else:
            ttt = computer_move(ttt, verbose=False)

        # Now check status
        status = ttt.get_status()
        print(status)
        if status != 'in progress':
            game_over = True
            print('Game over')

game()
```

Rubric.

Problem 1 total: 15 points

General deductions:

1. **(-1 points)** Code needs slight modification to compile
2. **(-2 points)** Code not submitted as separate file but easy enough to set up
3. **(-5 points)** Code not submitted as separate file and requires nontrivial collection or modification to compile, or does not compile

Problem 1a (2 points)

Follows problem convention

1. `Vertex` class contains current state (board values, whose turn)
2. `Vertex` contains adjacency method to track children (possible next positions) (`get_children`)

1. 0 for blank, 1 for X, -1 for O
2. 3×3 array

Problem 1b (2 points)

Follows problem convention

1. `Vertex` class contains status method (`get_status`) that returns if game is in progress or who winner is
2. `Vertex` contains adjacency method to track children (possible next positions) (`get_children`)

Problem 1c (5 points)

(1 point) (up to 4) per case (correct or not) that includes at least two of proposed position, number of winning outcomes, and total possible outcomes (`computer_move`)

(1 points) for including code with reasonable attempt

1. **(-1 point)** if 2+ proposed positions are missing/wrong
2. **(-1 point)** if 2+ number of winning outcomes are missing/wrong
3. **(-1 point)** if 2+ total possible outcomes are missing/wrong
4. **(-1 point)** if number of winning outcomes and total possible outcomes are missing but all win:total ratios are correct

Problem 1d (6 points)

(6 points) if game interface (`game`) appears, **(1 points)** if no interface but includes code with partial attempt, **(2 points)** if no interface but includes code that is close to working.

For an interface that runs,

1. **(-0 points)** if one of the below doesn't work
2. **(-1 point)** if two of the below are missing/work incorrectly
3. **(-2 points)** if three of the below are missing/work incorrectly
4. **(-3 points)** if four of the below are missing/work incorrectly
   (a) Human input works
   (b) Computer responds with a move
   (c) Displays board
   (d) Random player starts first or user selects starting player
   (e) Displays game status (in progress or winner) at each step
   (f) Indicates winner at end
   (g) Computer plays correct marker (X/O)
   (h) Computer plays into empty space
   (i) Cannot play in filled space

Problem 2

Part a. Here, we need to solve the coupled equations $\underline{\mathbf{f}} = \underline{\mathbf{0}}$, using a numerical solver.

$$0 = \frac{C_{\mathrm{A,inlet}} - C_{\mathrm{A}}}{\tau} - k\, C_{\mathrm{A}} \equiv f_1 \tag{1}$$

$$0 = \frac{T_{\mathrm{inlet}} - T}{\tau} - \frac{H_{\mathrm{rxn}}}{\rho\, c_p} k\, C_{\mathrm{A}} \equiv f_2 \tag{2}$$

This results in three unique solutions:

C_A, mol/L	T, K
1.905	301.9
0.807	323.9
0.253	335.0

Part b. We need to find the Jacobian, which in this case is the derivative of our two algebraic equations with respect to the two variables $x_1 = C_{\mathrm{A}}$ and $x_2 = T$.

$$f_1 = \frac{C_{\mathrm{A,inlet}} - C_{\mathrm{A}}}{\tau} - Ae^{-E_{\mathrm{A}}/RT} C_{\mathrm{A}}$$

$$f_2 = \frac{T_{\mathrm{inlet}} - T}{\tau} - \frac{H_{\mathrm{rxn}}}{\rho c_p} Ae^{-E_{\mathrm{A}}/RT} C_{\mathrm{A}}$$

$$J_{11} \equiv \frac{\partial f_1}{\partial x_1} = -\frac{1}{\tau} - Ae^{-E_{\mathrm{A}}/RT}$$

$$J_{12} \equiv \frac{\partial f_1}{\partial x_2} = -\frac{AE_{\mathrm{A}}C_{\mathrm{A}}}{RT^2} e^{-E_{\mathrm{A}}/RT}$$

$$J_{21} \equiv \frac{\partial f_2}{\partial x_1} = -\frac{H_{\mathrm{rxn}} A}{\rho c_p} e^{-E_{\mathrm{A}}/RT}$$

$$J_{22} \equiv \frac{\partial f_2}{\partial x_2} = -\frac{1}{\tau} - \frac{H_{\mathrm{rxn}} AC_{\mathrm{A}}E_{\mathrm{A}}}{\rho c_p RT^2} e^{-E_{\mathrm{A}}/RT}$$

Then we solve $\det \underline{\underline{\mathbf{J}}} = 0$ along with $\underline{\mathbf{f}} = \underline{\mathbf{0}}$. It's a little to find good initial guesses, but with enough playing around we get:

T_0	$C_{\mathrm{A,SS}}$	T_{SS}
304.0	1.60	312.1
298.8	0.47	329.5

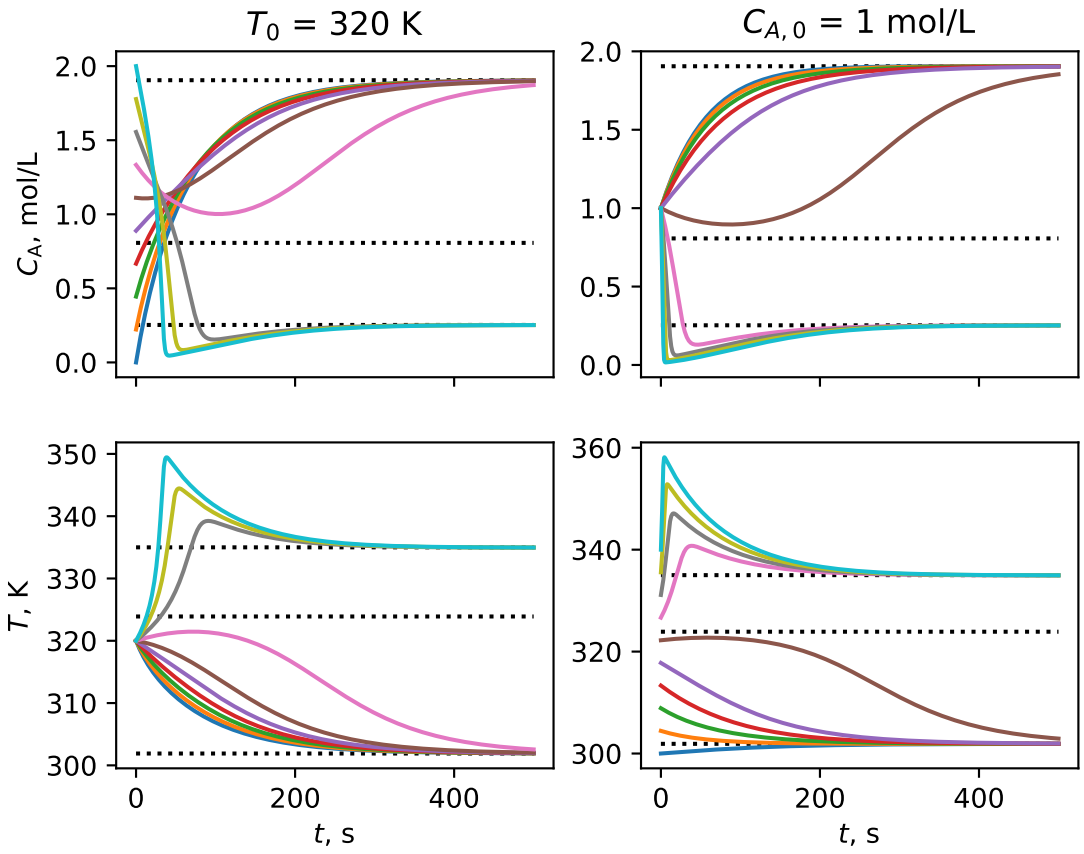
(Note that another good strategy is to just repeat part a, for various values of T_0; that is, use the SS values as the initial guess when solving for T_0 and see where the SS's converge into a single point. Other strategies are also acceptable.)

Part c. This part is now simple. We use the Jacobian we just found, and plug in our steady state values from part a (with $T_{\text{inlet}} = 300$ K), and find the eigenvalues. This gives:

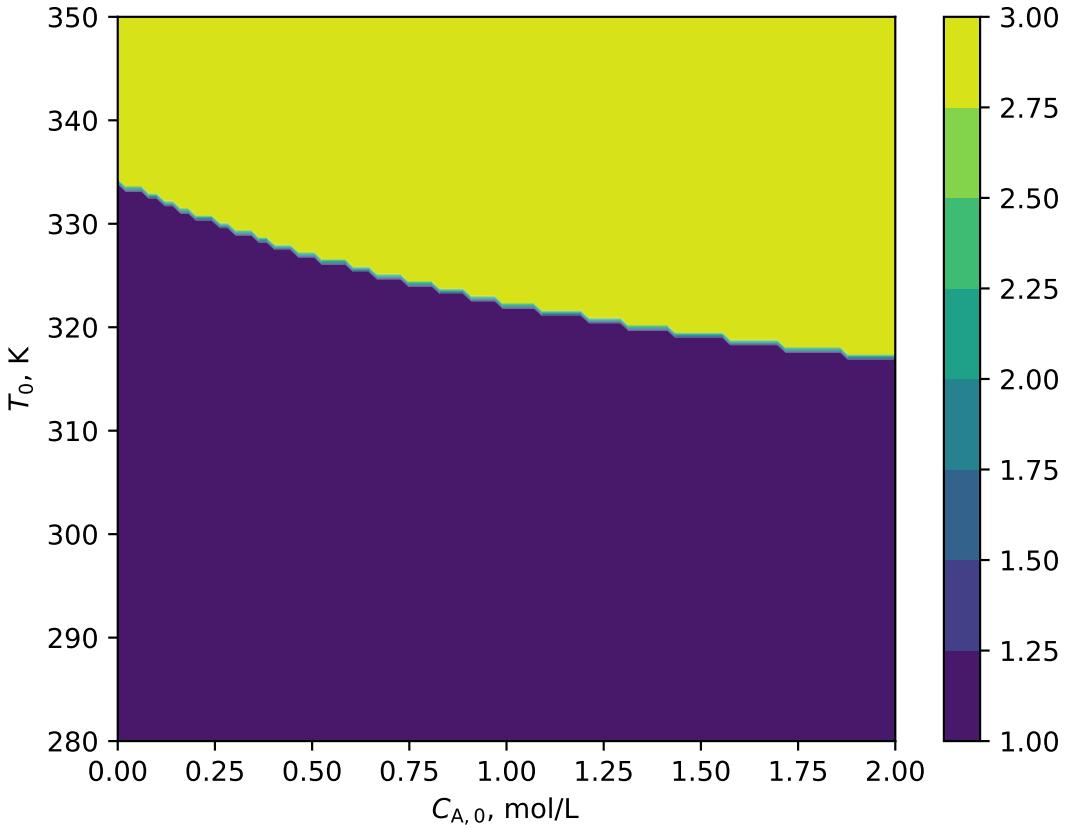
C_A	T	λ_1	λ_2	conclusion
1.91	301.90	-0.02	-0.01	stable
0.81	323.90	-0.02	0.02	unstable
0.25	335.00	-0.05	-0.02	stable

Part d. It is not stiff—the ratio of the eigenvalues is not large at any of the values we examined in the previous part.

Part e. Some trajectories are shown below; we see it always avoids the middle steady state.



Part f. Here, we just repeat the previous part at enough initial conditions to make a map, tracking which steady state the system ultimately arrives at. This gives.



Rubric. Total 20 points.

- Part a: 4 points

  -1 (repeated) missing any solution when others are present

- Part b: 4 points

  +2/4 right approach, but did not reach numerical answer
  +0/4 wrong answer, no work shown
  +1/4 described a seemingly valid approach

- Part c: 3 points

  +2/3 right approach; wrong result
  +0/3 only J shown
  -1 says it oscillates, even thoug Im(lambda)=0
  -1 incorrectly labels stable/unstable contradicting lambda signs

- Part d: 3 points

  +1/3 good reason, no numbers and wrong result
  +0/3 bad reasoning
  +3/3 good reasoning, but wrong result due to bad lambdas from previous

+1/3 right answer, but no apparent reasoning
+2/3 mostly right, but condition numbers clearly wrong
+3/3 all right, but judged 3 ¿¿ 1 (this is not stiff)

- Part e: 3 points

+1/3 trajectories only approach/diverge from 1 SS
+2/3 trajectories only approach/diverge from 2 SS
+1/3 well-described, but no plots shown
+1/3 such a huge range that SS's can't be distinguished
-1 only T or C shown, not both
-1 Time scale too short to reach SS

- Part f: 3 points

-2 moderately unclear phase diagram, more than one attribute wrong/missing
-1 showed region with unstable root (probably by contour interpolation)
-1 looks right, but seems to have swapped x and y?
-1 small irregularities on the shown plot, mostly right
0/3 missing/ very unclear phase diagram

Problem 3

Part a. Some example plots are shown below in Figure 1. At σ value of 0.001, we don't really see the noise, by 0.01 we are starting to see it. At 1, the signal is getting weak, and at 3 we can still see some hints of the color shading. By 10 it is basically lost to the noise.

part b. The known local minima are

x_1	x_2	f
-3.63	2.91	-12.70
2.98	2.32	-4.11
3.44	-3.10	-12.17
-3.26	-2.64	-9.73

The trickiest part of this part is determining a systematic means to say when your optimizer has "converged" to a solution. Figure 2 shows an example optimization; on the bottom the black points are the function value at that point. By eye (only looking at the points) we can see that it's converged to something around -14. However, we need to run several thousand of these simulations, so we need a way for our computer to figure out if it converged, and what it converged to. Many approaches are possible, here we show one simple example.

1. **Smoothing.** We need some way to smooth out the noise. Here, we'll use a cumulative running median; we can see this is better than a cumulative running average in this case as it is not so influenced by the one-sided outliers.

2. **When did it converge?** If we examine the cumulative running median (CRM), we need a way to say when it has converged. Here, we did this by comparing the current value of the CRM to the last n values, if all of the past n values were within ϵ_{tol} of the last value, we say it's converged. We found this worked reasonably well choosing $n = 10$ and $\epsilon_{\text{tol}} = 0.01$.

With the above, we can say what values our algorithm converged to on each iteration. Then, we can just compare this to our known local minima. Here, we find it better to compare the x_1, x_2 coordinates than the function value. We'll just use a cutoff of 0.5 for each. Then it's just a matter of letting the computer run and collecting statistics. Our results are shown in the table below, but student's results will vary depending upon their implementation. We see that our algorithm is actually pretty good until about $\sigma = 1$, which is also when we see the noise start to wash out the signal on our contour plots. Here are our statistics out of 1000 runs:

σ	Found same minima (%)	Found any minima (%)
0.001	100.0	100.0
0.010	100.0	100.0
0.100	100.0	100.0
1.000	39.4	74.6
3.000	3.4	12.9
10.000	0.4	1.6

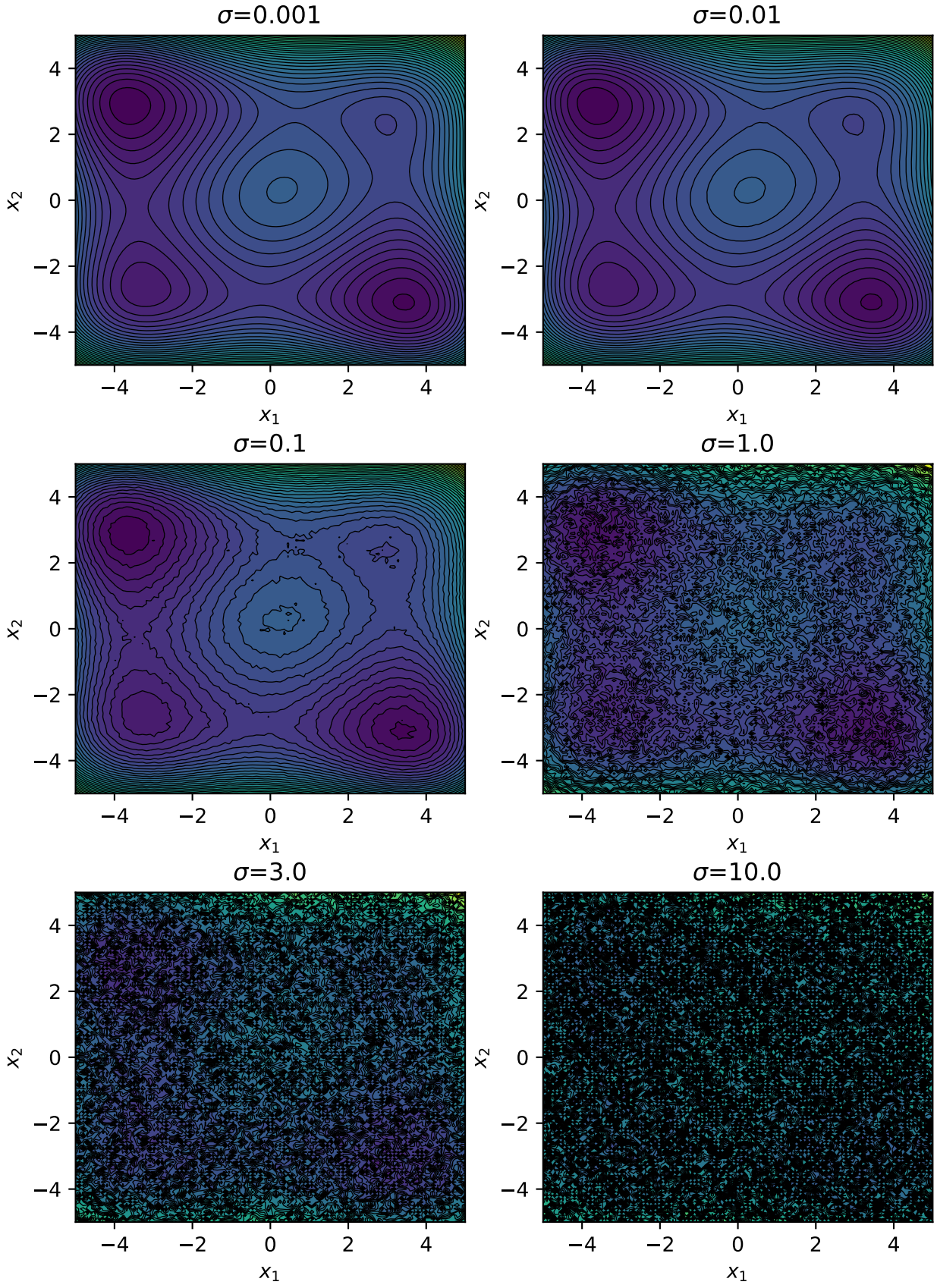


Figure 1: Contour plots with varying levels of noise.

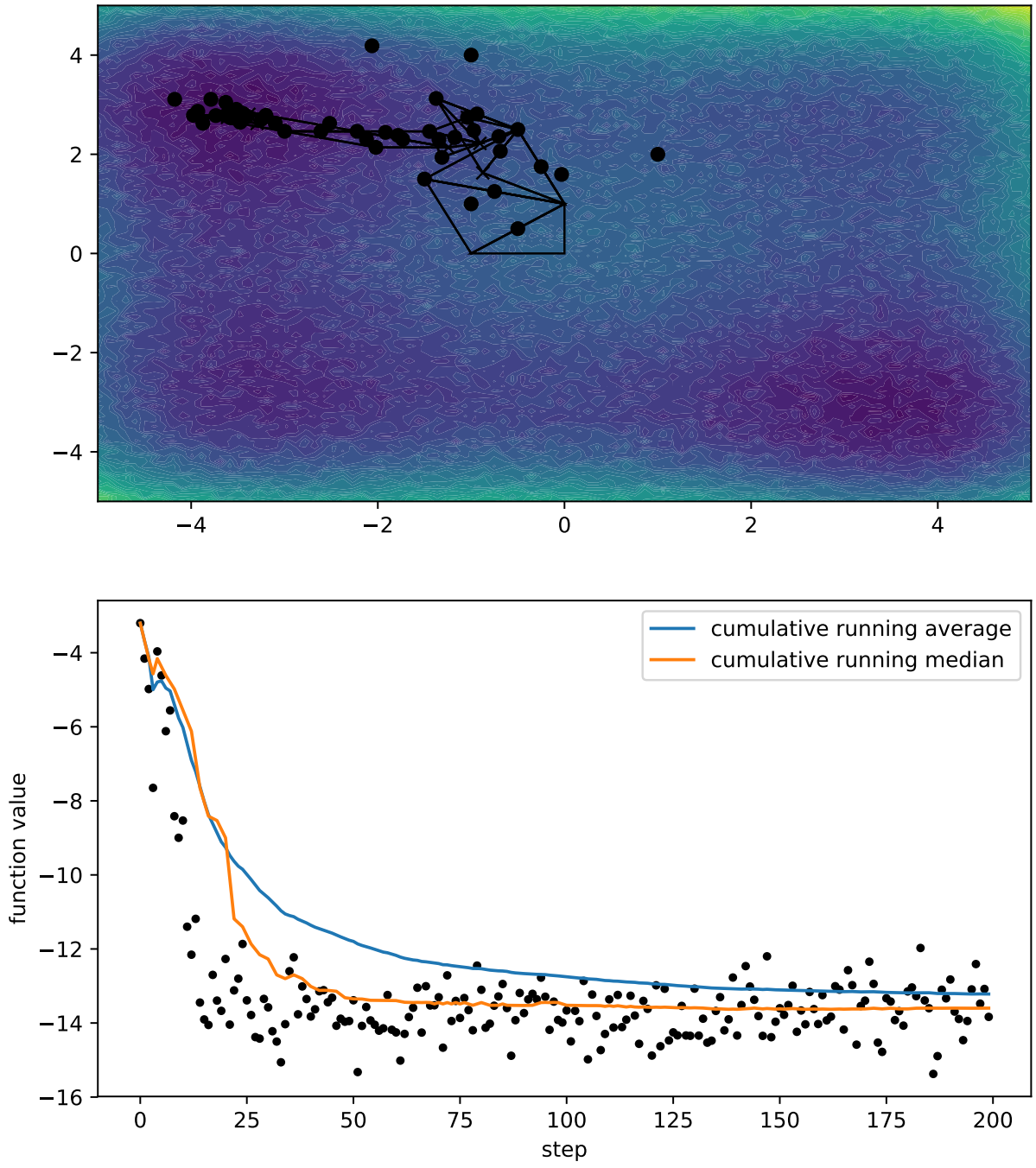


Figure 2: Example optimization.

Part c. The procedure is the same, but we let the algorithm draw its own conclusions about when it has converged. It turns out our homemade algorithm is better for this purpose! Statistics are below.

Nelder–Mead.

σ	Found same minima (%)	Found any minima (%)
1×10^{-3}	99.9	100.0
3×10^{-3}	79.1	94.9
1×10^{-2}	27.9	63.1
3×10^{-2}	2.3	3.2
1×10^{-1}	0.0	0.0

BFGS.

σ	Found same minima (%)	Found any minima (%)
1×10^{-9}	100.0	100.0
3×10^{-9}	95.3	98.4
1×10^{-8}	62.6	77.0
3×10^{-8}	18.0	27.2
1×10^{-7}	4.0	7.4

Rubric

- Part a (4 points)
  - +4 correct plots with at least three values, including where noise starts significant and where washes out the function
  - +3 correct plots with 2 values, including where noise starts significant and where washes out the function
  - +2 correct plots with 1-2 values
  - +1 incorrect plots but correct attempt or code makes sense
  - +0 incorrect plots with incorrect attempt or no code
- Part b (7 points)
  3 points:
  - +3 reasonable criteria for finding the minimum of noisy function and 1-2 sample plots
  - +2 reasonable criteria for finding the minimum of noisy function without sample plots
  - +1 wrong criteria or criteria doesnt make sense.
  - +0 no/wrong criteria or sample plots provided.

  2 points:
  - +2 at least five representative values have been tested and report the number of times it converges to the same local minimum.

+1 1-4 representative values have been tested and report the number of times it converges to the same local minimum.

+1 representative values have been tested but didnt report the number of times it converges to the same local minimum.

+1 report the number of times it converges but values are not representative.

+0 no work/results presented

2 points:

+2 at least five representative values have been tested and report the number of times it converges to any local minimum

+1 1-4 representative values have been tested and report the number of times it converges to any local minimum

+1 representative values have been tested but didnt report the number of times it converges to any local minimum

+1 report the number of times it converges but values are not representative.

+0 no work/results presented

Extra:

+1 If no results presented in above two sections, but code provided

- Part c (4 points)
  2 points for Nelder-Mead

  +2 at least five representative values have been tested for same local minimum and any local minimum and report the number of times it converges

  +1 at least five representative values have been tested for same local minimum or any local minimum but didnt report the number of times it converges

  +1 1-4 representative values have been tested for same local minimum and any local minimum

  +1 report the number of times it converges but values are not representative.

  +0.5 wrong results but code provided

  +0 no work/results presented

  2 points for BFGS

  +2 at least five representative values have been tested for same local minimum and any local minimum and report the number of times it converges

  +1 at least five representative values have been tested for same local minimum or any local minimum but didnt report the number of times it converges

  +1 1-4 representative values have been tested for same local minimum and any local minimum

  +1 report the number of times it converges but values are not representative.

  +1 no/wrong results but good reasoning

  +0.5 no/wrong results but code provided

  +0 no work/results presented

Late penalties

Late penalties were assigned as follows:

- A grace period of 30 minutes was given; that is no penalty if the assignment was <30 min late.
- 5% penalty per 30 minutes thereafter.